VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

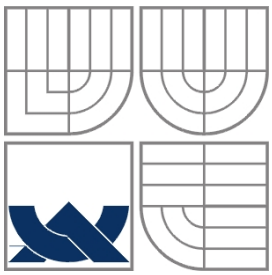
SYSTÉM LOGOVÁNÍ ZPRÁV

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

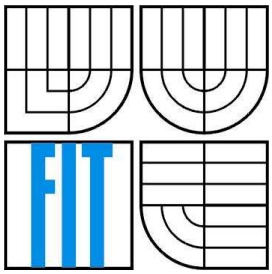
AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV VRZAL

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM LOGOVÁNÍ ZPRÁV

MESSAGE LOGGING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MIROSLAV VRZAL
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2009

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2008/2009

Zadání diplomové práce

Řešitel: **Vrzal Miroslav, Bc.**
Obor: Informační systémy
Téma: **Systém logování zpráv**
Kategorie: Operační systémy

Pokyny:

1. Analýzujte systém zpráv v operačním systému IBM i5/OS (AS/400, iSeries, i5), případně v dalších operačních systémech.
2. Analýzujte možností systémů logování zpráv nezávislých na operačním systému se zaměřením na systémy, které podporují Javu.
3. Definujte vlastností obecného systému logování zpráv nezávislého na operačním systému.
4. Implementujte systém logování zpráv v jazyku Java který bude umožňovat definici zprávy, zasílání zpráv a další operace se zprávami podrobněji specifikované v zadání poskytnutém zadavatelem.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj systému.

Literatura:

- Holt, T., Malaga, E.: Complete CL: The Definitive Control Language Programming Guide, Fourth Edition, Mc Press, 2004.
- iSeries Information Center. Infromace dostupne na http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp.
- Bauer, Ch., King, G.: Java Persistence with Hibernate, Manning, 2006.
- Machacek, J., Ditt, J., Vukotic, A., Chakraborty, A.: Pro Spring 2.5, Apress, 2008.
- Clayber, E., Rubel D.: Eclipse: Building Commercial-Quality Plug-ins, Addison-Wesley, 2006.

Při obhajobě semestrální části diplomového projektu je požadováno:

- 1 až 3

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Konzultant: Šenk Zdeněk, Ing., Aegis
Datum zadání: 22. září 2008
Datum odevzdání: 27. ledna 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato diplomová práce se zabývá analýzou systému IBM AS/400 a jeho systému zpráv. Na základě některých jeho principů je následně navržen a implementován v jazyce Java vlastní logovací systém pro logování aplikací pro firmu Aegis s.r.o. V první části práce popisuje systém AS/400 a jeho systém zpráv a soustředí se především na následující oblasti: předdefinování zpráv a jejich uložení, typy zpráv, úrovně závažnosti zpráv, možnost práce s proměnnými v textu zpráv a způsob zasílání zpráv. Součástí práce je také analýza dalších systémů logování zpráv, přičemž jsou zde popsány syslog a syslog-ng používané na UNIXových systémech. Soustřeďuje se na: typy zpráv, úrovně závažností zpráv a možností filtrování zpráv a jejich ukládání. Dále pak popisuje utilitu Log4j pro logování aplikací v Javě. V druhé části se práce věnuje návrhu a implementaci vlastního logovacího systému.

Abstract

This master's thesis in the first part describes the AS/400 and its message system and concentrates especially on the following areas: predefinition of messages and their storing, types of messages and levels of their importance, work with variables included in message text and ways of sending messages. On the basis of AS/400 message system is designed and implemented message log system for the application loggin for Aegis. s.r.o. The analysis of the message log systems is also a part of the work. The syslog and syslog-ng used in UNIX systems are described, concerning types of messages, importance of messages and filtering and storing of messages. It further describes possibilities of application logging based on *Java* in the specific case of the Log4j utility. In the second part thesis describes own log message systems design and implementation.

Klíčová slova

Logování, log, logovací zpráva, aplikační logování, operační systém, AS/400, syslog, syslog-ng, Java, Log4j, databáze, UML, JDBC, Spring.

Keywords

Logging, log, log massage, application logging, operating system, AS/400, syslog, syslog-ng, Java, Log4j, databáze, UML, JDBC, Spring.

Citace

Miroslav Vrzal: Systém logování zpráv, diplomová práce, Brno, FIT VUT v Brně, 2009.

Systém logování zpráv

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Jaroslava Zendulky, CSc.
Další informace a technickou podporu mi poskytli:
Ing. Zdeněk Šenk konzultant firmy Aegis s.r.o. pro tuto diplomovou práci
Ing. Tomáš Vítek zaměstnanec firmy Aegis s.r.o.
Evžen Kučera majitel firmy Aegis s.r.o.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Miroslav Vrzal
3.2.2009

Poděkování

Chtěl bych na tomto místě poděkovat doc. Ing. Jaroslavu Zendulkovi, CSc., Ing. Zdeňku Šenkovi, Ing. Tomáši Vítkovi a Evženu Kučerovi za odborné vedení a konzultace, které mi poskytl v teoretické i implementační části diplomové práce.

Obsah

1 Úvod

Tato diplomová práce je vytvářena na základě zadání od firmy Aegis s.r.o. Práce ve své první části popisuje IBM AS/400 a jeho systém zpráv. Soustředí se především na následující oblasti: předdefinování zpráv a jejich uložení, typy zpráv, úrovně závažnosti zpráv, možnost práce s proměnnými v textu zpráv a způsob zasílání zpráv.

Součástí práce je také analýza logování zpráv dalších systémů – jsou zde popsány syslog a syslog-ng, používané na UNIXových systémech, a utilita Log4j pro logování v Javě, se zaměřením na: typy zpráv, úrovně závažností zpráv, možností filtrování zpráv a jejich ukládání.

V druhé části práce, po definování vlastností navrhovaného systému, následuje návrh a implementace logovacího systému v jazyce Java, který je určen pro logování aplikací pro firmu Aegis s.r.o. Tento systém podle požadavků firmy vychází z některých principů systému zpráv operačního systému IBM AS/400, avšak bude na operačním systému nezávislý. Použité principy a prvky ze systému zpráv na AS/400 budou především:

- Předdefinování zpráv a jejich externí uložení mimo samotný log v souborech zpráv (v našem případě v databázi).
- Parametrizace textu předdefinovaných zpráv s možností práce s proměnnými v textu zpráv.

Na závěr práce obsahuje popis možností dalšího rozvoje logovacího systému a shrnutí dosažených výsledků.

V příloze práce jsou pak uvedeny výsledky jednotlivých testů systému a popis jeho konfigurace.

Obsahem diplomové práce je 11 kapitol. Každá z těchto kapitol se zabývá určitým tématem.

- Nejprve v kapitole 2 Aegis s.r.o. uvedeme v krátkosti základní informace o firmě Aegis s.r.o. Tedy informace o zaměření a činnosti firmy, sídle firmy, kontaktu, a také uvedeme důležité osoby, které měly na tvorbu práce (ať už odbornými konzultacemi, školením, poskytnutím potřebných softwarových a hardwarových prostředků či jiným způsobem) vliv.

- V kapitole 3 Logování zpráv vymezíme obecně pojem logování zpráv, popíšeme hlavní důvody k logování událostí a také uvedeme možnosti ukládání logovacích zpráv.

- V další kapitole 4 IBM AS/400 se nejprve budeme zabývat charakteristikou samotného systému AS/400. Popíšeme jednotlivé rysy systému AS/400, jeho jednotlivé důležité prvky a celkový kontext, ve kterém systém zpráv, jemuž se budeme v této kapitole následně věnovat, pracuje. Zaměříme se především na oblasti: předdefinování zpráv a jejich uložení, typy zpráv, úrovně závažnosti zpráv, možnost práce s proměnnými v textu zpráv a způsob zasílání zpráv.

- V další kapitole 5 Analýza dalších vybraných logovacích systémů se budeme stručně zabývat konkrétními existujícími systémy pro logování – syslog a syslog-ng. Konkrétně se zaměříme na analýzu aspektů, které jsou důležité pro návrh našeho vlastního logovacího systému, tedy formát zprávy, definování zdrojů a cílů zpráv, možnost konfigurace, úrovně závažnosti, možnosti filtrace logu a způsob jeho ukládání.

- Kapitola 6 Logování ve spojení s Javou se zabývá možností logování zpráv z aplikací. Konkrétně logováním v aplikacích na bázi programovacího jazyka Java. Bude zde stručně popsán příklad systému pro logování aplikací v Jave – utilita Log4j.

- V kapitole 7 Definice vlastností obecného logovacího systému za použití principů z AS/400 jsou na základě analogie některých principů ze systému zpráv v AS/400 obecně definovány jednotlivé vlastnosti logovacího systému, které by navrhovaný logovací systém měl splňovat.

- V další kapitole 8 Analýza požadavků a návrh systému je vytvořen popis systému a specifikace konkrétních požadavků zadavatele na logovací systém. Je zde uveden diagram použití a vytvořen návrh struktury databáze a objektů pro perzistentní a aplikační vrstvu systému.

- Kapitola 9 Implementace a testování aplikace již obsahuje, spolu se stručným popisem použitých nástrojů, popis konkrétních postupů při realizaci systému. Je zde také uvedeno jakým způsobem byly provedeny testy systému a demonstrativně předloženy konkrétní výstupy (sejmuté obrazovky) vytvořené při práci se záznamy v logu a předdefinovanými zprávami.

- Předposlední kapitola 10 Možnosti rozšíření obsahuje nástin několika možností dalšího rozvoje systému.

- Poslední kapitolou je 11 Závěr, kde se nachází shrnutí práce a zhodnocení dosažených výsledků.

2 Aegis s.r.o.

Aegis s.r.o je společnost zaměřená na vývoj software a školení v oblasti technologií IBM, která působí v oblasti IT služeb od roku 1997.

Vedení společnosti:

Evžen Kučera - *ředitel a jednatel společnosti*
Richard Strouhal - *vedoucí projektů a jednatel společnosti*

Sídlo firmy:

Na Pankráci 58, 140 00 Praha 4
IČ: 25621564
DIČ: CZ25621564

Pobočka Brno:

Polní 5, 639 00 Brno

www stránky:

http://www.aegis.cz/

Konzultant v rámci této diplomové práce: Ing. Zdeněk Šenk.

Školení pro práci v *AS/400*: Evžen Kučera.

Technická podpora a konzultace v implementeční části: Ing. Tomáš Vítek.

3 Logování zpráv

V této kapitole se budeme obecně věnovat vymezení logování zpráv. Jako zdroje informací byly použity materiály [5], [10], [11], [13], [21], [28], [30].

3.1 Logovací zpráva a log

Logovací zpráva je textové sdělení reprezentující popis nějaké konkrétní události, která nastala v operačním systému, aplikaci, na zařízení atd. Logovací zprávy jsou postupně společně s časem, kdy ke vzniku události došlo, a případně dalšími parametry, zaznamenávány a ukládány. Záznam logovacích zpráv (nazývaný též jako log) poskytuje v časovém sledu historii událostí spojených s jednotlivými objekty (aplikacemi, uživateli, zařízeními atd.), které se na vzniku událostí a generování logovacích zpráv podíleli.

Vytváření zpráv, jejich ukládání a zpracování zajišťují systémy logování zpráv, které používají nejen operační systémy, ale i jednotlivé softwarové projekty mají často subsystémy zajišťující sledování aplikací a vytváření zpráv o událostech, které v nich nastaly.

3.2 Důvody k logování zpráv

Důvody pro provádění logování by se daly rozdělit zhruba do následujících oblastí:

3.2.1 Zvýšení bezpečnosti systému a odhalování útoků

Logování zpráv je jedním z důležitých bezpečnostních mechanismů, protože ze záznamů v logu je možné odhalit průběh útoků (někdy je možné odhalit i útok v reálném čase a reagovat na něj příslušnými bezpečnostními opatřeními), způsob jakým byly útoky provedeny, odkud, kdy a kdo je provedl. Následně lze podle záznamů zjistit slabá místa systému a vyvodit odpovědnost za způsobené škody, napomoci při obnově aplikace do původního stavu a případně se i dopátrat útočníka. Záznamy logu mohou hrát důležitou roli při provádění bezpečnostního auditu. Ten může napomoci ke stanovení bezpečnostních hrozeb a následné implementaci bezpečnostních opatření.

Záznamy logu se však samy mohou stát cílem útočníka a mohou být neoprávněně modifikovány nebo smazány. Útočník se může snažit skrýt svoji identitu a odstranit z logu události, které s útokem souvisely, proto by na důležité záznamy logu měla být aplikována příslušná bezpečnostní opatření. Záznamy logu by měly být uloženy na bezpečném místě a zálohovány, případně také zašifrovány. Přístup k logu by měla mít pouze autorizovaná osoba.

3.2.2 Analýza fungování aplikací pro detekci chyb a vylepšení aplikací

Díky zapisování chybových zpráv do logu lze detekovat chyby programů a hardwaru a zjistit kontext za jakých podmínek a kdy tyto chyby vznikají.

Na základě zpráv lze také analyzovat operace uživatele v aplikaci. Například četnost využívání jednotlivých implementovaných funkcí, nebo cestu uživatele v aplikaci – posloupnost jeho operací v čase. Analýza logu tak může vést k následnému vylepšování aplikací. Například vyřazení nepoužívaných funkcí nebo úpravu prostředí pro zlepšení orientace v aplikaci pokud zjistíme, že uživatel potřebnou funkci dlouho hledá.

3.2.3 Sledování chování uživatelů pro vylepšení obchodní strategie

Pomocí záznamu v logu lze sledovat chování jednotlivých uživatelů v internetových obchodech. Na základě zjištěných poznatků o chování uživatele (například nejčastěji prohlížená kategorie zboží a podobně) a jeho zvyklostech (typicky volené operace v navigaci obchodu atd.), lze nastavit každému zákazníkovi prostředí obchodu tak, aby lépe vyhovovalo jeho požadavkům a přáním. V rámci jeho typických zvyklostí a chování mu pak může být nastavena optimální navigace a nabídnuto zboží, které ho podle četnosti prohlížení daných kategorií nejvíce zajímá, a zvýšit tak možnost jeho nákupu a tím i potencionální zisk provozovatele obchodu.

3.3 Obsah logované zprávy

Každá událost, která nastala má konkrétní příčiny a kontext, ve kterém vznikla. Logovací zpráva obsahuje textový popis této události, kde mohou být uvedeny příčiny, kontext a případně i následky dané události.

Události nastávají v určitých časových okamžicích, proto je dalším důležitým parametrem logovací zprávy datum a čas[1], kdy k události došlo. Dále mohou být obsahem logu další informace jako kódové označení a typ události, její závažnost, identifikace zdroje zprávy (IP adresy počítačů, jméno počítače, aplikace a podobně.) a identifikace objektů, které se na vzniku události podílely – názvy aplikací, jména souborů, identifikace jednotlivých uživatelů atd.

Formát a obsah logu může být velmi různorodý a záleží na konkrétních aplikacích, respektive jejich tvůrcích a administrátorech systémů, které informace pokládají za důležité a v jakém formátu je chtějí do logu ukládat. Záznam skutečného logu může vypadat například následujícím způsobem:

```
[05/Apr/2008:17:12:02 +0200] "GET
/csp/user/person.csp?jmeno=Miroslav&prijmeni=Vrzal&vek=26 HTTP/1.1"
```

[1] Existují různé formáty zápisu datumu. Například DD-MM-YYYY nebo MM-DD-YYYY (tzv. americký formát). Y – rok; M – měsíc, D – den.

3.4 Ukládání a správa logovacích záznamů

Záznamy logu mohou obsahovat důležité informace o aplikacích, nebo například i důvěrné informace o uživatelích (hesla, rodná čísla, čísla účtů atd.), proto by k nim měla mít přístup jen oprávněná osoba. Získání logu neoprávněnou osobou zvyšuje možnost odhalení slabých míst systému a následných útokům na systém, nebo hrozbu případných finančních ztrát či trestního postihu při zveřejnění osobních údajů osob.

Pro správu logu, respektive pro výběr zpráv pouze určitého typu, bývá zpravidla aplikován nějaký filtrační mechanismus, který na základě stanovených kritérií vybírá pouze některé zprávy. Kritéria typicky zahrnují čas, typ události a entitu, která událost způsobila.

Záznamy logu je třeba ukládat a v pravidelných intervalech zálohovat. V zásadě existují dvě používané varianty ukládání záznamů logu: ukládání do logovacích souborů, a nebo ukládání logu do databáze.

3.4.1 Ukládání logu do souborů

Při ukládání logů do souborů jsou nové záznamy přidávány na konec souboru. U důležitých logů jsou přístupová práva k těmto souborům nastavována pro čtení a zápis pouze příslušným uživatelům s patřičnou úrovní oprávnění.

Soubory logu mají omezenou velikost, proto je při naplnění souboru potřeba buď vytvořit soubor nový (lze například vytvářet adresáře a soubory podle časového období, podle logované aplikace a podobně), nebo nejstarší záznamy odstraňovat (tzv. ořezávání logu) a nahrazovat novými.

Aby logovací soubory nezabíraly příliš místa na disku, je možné provádět tzv. rotaci logu. To znamená, že původní soubor je zkomprimován, dostane nový název a log je do původního souboru zapisován od začátku znovu.

Soubory logu jsou často obyčejné textové soubory (`.txt`) nebo soubory s příponou `.log`. Prohledávání logu uloženého v souborech je pak možno pomocí nástrojů, které vyhledávají podle klíčových slov v textu.

3.4.2 Ukládání logu do databáze

Metoda ukládání logovacích zpráv do souborů může být sice jednoduchá a rychlá, ale neposkytuje příliš možností pro další práci s logem, jako je například vyhledávání nebo vizualizace vybraných zpráv.

To nám lépe umožňuje log uložený v databázi, protože nad jednotlivými záznamy v databázi lze provádět pomocí příkazů pro práci s databází dotazování, při kterém lze zadat konkrétní kritéria (například časový interval, závažnost, identifikace aplikací, uživatelů, procesů a podobně) a v logu efektivněji vyhledávat a případně data také modifikovat. Je také možné je dále předávat aplikacím, které data vizualizují nebo provádí jejich další analýzu.

4 IBM AS/400

Před samotnou analýzou systému zpráv na AS/400 je nejprve důležité uvést základní charakteristiky a principy fungování AS/400, do kterých je systém zpráv zasazen. Těmto principům a charakteristikám se budeme věnovat v následujících kapitolách, přičemž jako zdroj informací byly použity materiály [1], [2], [3], [4], [8], [12], [13], [14], [15], [17], [32].

4.1 Historie AS/400

V roce 1988 byl IBM uveden počítač s názvem AS/400 vyvinutý laboratořích IBM v americkém Rochesteru jako následník počítače System/38. Tento počítač disponoval poměrně netradiční architekturou, zcela odlišnou od ostatních systémů, vytvořenou Frank G. Soltisem, který byl za jeho vývoj zodpovědný.

AS/400 začal být především pro svou bezpečnost[2] a spolehlivost[3] využíván zejména pro obchodní aplikace jako databázový server. Jako první všestranně použitelný počítač dosáhl také hodnocení bezpečnosti podle kritérií TCSEC[4] úrovně C2[5].

AS/400 prošel ze strany IBM postupně několika změnami v označení[6] (v rámci této práce se však pro přehlednost budeme držet původního a stále používaného označení AS/400). V roce 2000 došlo u AS/400 k přejmenování na eServer iSeries, které bylo v roce 2006 změněno na IBM System i a v roce 2008 na IBM Power system, což zahrnovalo jeho integraci spolu s platformou IBM System p[7].

Také operační systém vytvořený pro AS/400 doznal změnu v názvu. Původní OS/400 nahradilo i5/OS označované také jako IBM i.

4.2 Charakteristika AS/400

Jak již bylo řečeno, architektura AS/400 je poměrně odlišná od koncepcí ostatních výpočetních systémů. Samotná firma IBM (zdroj [15]) prezentuje jako pět klíčových principů AS/400 tyto aspekty:

1) Technologickou nezávislost; 2) Objektově orientovanou strukturu; 3) Integraci hardwaru; 4) Integraci softwaru; 5) Jednoúrovňovou paměť.

Podrobněji se jednotlivými důležitými vlastnostmi AS/400 budeme zabývat v dalších kapitolách.

---

[2] Počítačové viry se zde prakticky nevyskytují, protože je není možné, díky předdefinovaným pravidlům pro práci s objekty, napsat.

[3] Po optimálním nastavení systém může bez závady pracovat i řadu let.

[4] *Trusted Computer Systems Evaluation Criteria*. Kritéria amerického ministerstva obrany definovaná v tzv. „*Orange book*“.

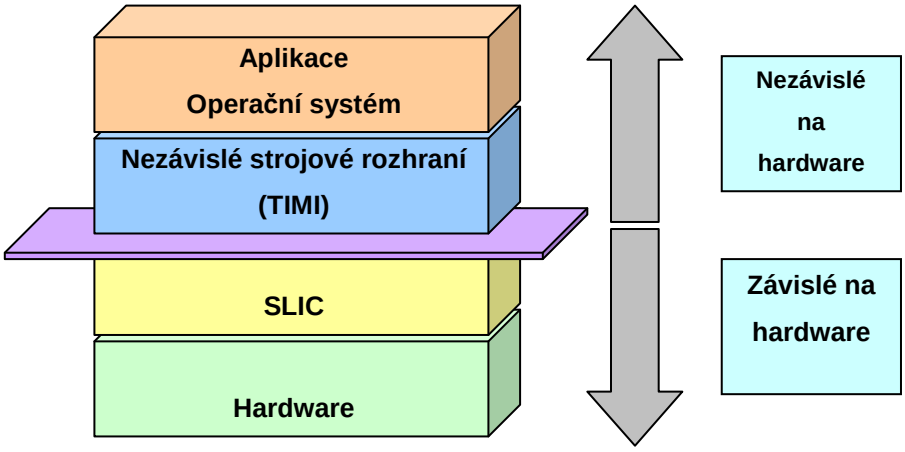
[5] Ochrana s řízeným přístupem zahrnujícím audit činností.

[6] Se změnami označení AS/400 ze strany IBM lze pravděpodobně počítat (například i z marketingových důvodů) i do budoucna.

[7] System p zahrnuje servery podporující operační systémy AIX a Linux.

4.2.1 Technologická nezávislost (nezávislé strojové rozhraní)

Na rozdíl od jiných počítačových systémů není architektura AS/400 definovaná hardwarem, protože programy nekomunikují s hardwarem přímo, ale pouze s rozhraním TIMI (*Technology Independent Machine Interface*). TIMI obsahuje tzv. TIMI instrukce. TIMI instrukce nejsou přímo proveditelné, jsou pouze virtuální instrukční sadou. Ve finálním kompilačním kroku jsou tyto instrukce přeložené do instrukcí procesoru. Mezi vrstvou TIMI a hardwarem se nachází systémová softwarová vrstva SLIC (*System Licensed Internal Code*), což je mikrokódová úroveň vytvořená IBM ke konkrétnímu hardwaru. Při změně hardwaru tedy IBM pouze pozmění kód ve vrstvě SLIC a vrstva TIMI zůstává beze změny. Jednotlivé vrstvy AS/400 ilustruje následující obrázek.



Obr. č. 1: Vrstvy v systému AS/400

4.2.2 Jednoúrovňová paměť

Disková paměť AS/400 se jeví jako souvislý celek a veškerá paměť, jak operační, tak i disková, je adresována jednotně. Pokud je přidána nová paměťová komponenta, systém novou paměť sám rozpozná a začne ji automaticky používat.

O fyzické umístění objektů na paměťové médium a jejich ovládání se stará pouze operační systém, který si celý adresný prostor sám spravuje. Samotný uživatel nemusí znát žádné mechanismy ukládání dat a také se nemusí starat o to, kde jsou jednotlivé objekty uloženy. Ty bývají zpravidla rozloženy na více discích. Na jednotlivé objekty se pak uživatel odkazuje pouze jménem (nikoliv adresou).

4.2.3 Hardwarová integrace (procesory a I/O systém)

Pokud jsou data potřebná k provedení transakce uložená na pevném disku, odkud je třeba je načíst do paměti, předává hlavní procesor I/O požadavek načtení dat z disku procesoru, který je přidělený pro příslušný pevný disk. Hlavní procesor pak vykonává jiné činnost a k původní úloze se vrací až jsou potřebná data načtena v paměti. Tento princip je vhodný zvláště pro aplikace náročná na data, pro který je AS/400 primárně určen.

Původní AS/400 používaly (stejně jako System/38) 48-bitové CISC procesory. V roce 1995 přešla IBM u AS/400 na 64-bitovou RISC architekturu PowerPC, což vedlo k vytvoření řady

procesorů POWER (*Performance Optimization With Enhanced RISC)* a k výraznému zvýšení výkonu u výpočetně náročných operací. V současné době je nejnovější verzí POWER6 a v laboratořích IBM je již vyvíjen procesor POWER7.

4.2.4 Softwarová integrace (operační systém, licenční programy)

Operační systém je bezplatně dodáván (jako jeden z licenčních programů) společně s hardwarem AS/400. Funkce tohoto operačního systému zahrnuje především následující oblasti:

- Správa objektů (*Object management*)
- Správa dat (*Data management*)
- Řízení práce (*Work management*)
- Komunikace (*Communications*)
- Řídící jazyk (*Control language*)

Mnohé části těchto funkcí jsou realizovány ve vrstvě SLIC (například kontrola oprávnění k použití objektu, vytváření indexů k datům v databázovém souboru, obsluha zařízení atd.)

Licenční programy mohou být například různé překladače (např. AS/400 Pascal, COBOL 400, C++ atd.), kancelářské balíky (Office Vision 400), programy pro spolupráci s klienty na jiných platformách (Client Acces/400 – PC Support/400) atd.

Jak již bylo řečeno výše, nedílnou součástí AS/400 je také přímo do operačního systému integrovaný relační databázový systém DB2/400.

4.2.5 Objektově orientovaná struktura (objekty, knihovny)

AS/400 je na úrovni operačního systému označován jako plně objektově orientovaný systém [15]. Vše co je v paměti systému AS/400 uloženo je definováno jako objekt (výjimky tvoří systémové hodnoty, síťové atributy a tiskové soubory).

Na tomto místě by bylo vhodné říci, že objektově orientovaný návrh by měl obecně splňovat tato kritéria: zapouzdřenost, dědičnost a polymorfismus. V AS/400 však objekty nepodporují dědičnost. Proto by bylo v této souvislosti vhodnější hovořit o AS/400 spíše než o objektově orientovaném (*object oriented*) systému, jako na objektech založeném (*object based*) systému.

Objektově orientovaný návrh má významný vliv na bezpečnost. V AS/400 se prakticky nemohou vyskytovat počítačové viry[8]. AS/400 je tak systém s vysokou mírou bezpečnosti a integrity, což je důležité například pro finanční instituce, pro které je oblast bezpečnosti dat klíčová..

Objekt je obecně jakýsi uzavřený „kontejner“, který obsahuje datové struktury uživatele nebo systému. V AS/400 lze k těmto datům přistupovat pouze s příslušným oprávněním. V AS/400 je objekt pojmenovaná oblast paměti (každý objekt tedy musí mít jméno) a lze s ním provádět pouze předem definované operace pro práci s objekty. Tyto operace jsou prováděny pomocí množiny příkazů řídícího jazyka (*Control languague*). Podrobněji se řídícím jazykem budeme zabývat v kapitole 4.3.2 Control Languague.

[8] Pokud se virus snaží maskovat jako data, následně se přeměnit na programový kód a spustit svoji činnost, narazí na integritní omezení systému, protože taková změna charakteristiky není v systému AS/400 přípustná – když systém přijme soubor jako datový, nelze tuto charakteristiku již změnit. Není tedy možné, aby se virus spustil jako program.

Každý objekt v AS/400 má tyto atributy:

- jméno objektu
- typ objektu
- velikost objektu
- datum vytvoření objektu
- popis objektu vytvořený tvůrcem objektu

Jméno a typ objektu jsou pak používány k identifikaci objektu. Jméno objektu vytváří uživatel, který je zároveň také tvůrcem objektu. Typ objektu je definován příkazem, kterým byl objekt vytvořen (např. příkaz pro tvorbu knihovny vytvoří objekt typu knihovna).

Některé typy objektů v AS/400 předkládá následující seznam:

- Soubory
- Programy
- Příkazy
- Knihovny
- Fronty
- Moduly
- Servisní programy
- Uživatelské profily
- Popisy úloh
- Popisy subsystémů
- Popisy zařízení

Systém používá jméno a typ objektu automaticky pro jeho uložení a vykonávání operací s ním. Různé objekty mohou mít stejné jméno, ale každý musí mít jiný typ, nebo musí být uložen v jiné knihovně.

Knihovna je objekt typu `*LIBL` a slouží k seskupování objektů, které k sobě mají nějaký vztah. Knihovna je tedy adresář skupiny objektů. Knihovny v AS/400 nemají hierarchickou strukturu – jsou v jedné úrovni. To znamená, že knihovna může obsahovat objekty všech typů, ovšem mimo objektu typu knihovna. Výjimku tvoří systémová knihovna `QSYS`, ve které jsou uloženy všechny objekty typu `*LIB`. V AS/400 také existuje knihovna `QTEMP` pro dočasné ukládání pracovních objektů, která vzniká a zaniká s úlohou.

Při vytváření knihovny lze specifikovat paměťovou oblast ASP (*auxiliary storage pool*)[9], kde bude knihovna vytvořena. Všechny objekty, které jsou pak v této knihovně vytvořeny, se budou nacházet ve stejném ASP jako knihovna. Objekty v knihovně spolu nemusí nutně fyzicky sousedit. Systém sám najde potřebnou volnou paměť pro objekt, když je ukládán v systému.

Jméno knihovny může být také použito pro další úroveň identifikace jména objektu. Kombinace jména objektu a jména knihovny, ve které je objekt uložen, se nazývá kvalifikované jméno objektu.

Například: `POKUSLIB/POKUSPGM`

[9] ASP je paměťová oblast, která může zahrnovat několik konkrétních fyzických paměťových disků. Alespoň jedna ASP musí být systémová.

4.3 Práce s objekty v AS/400

4.3.1 Typy uživatelů a oprávnění k objektům

Bezpečnostní atributy jednotlivých uživatelů jsou nastaveny v samotném uživatelském profilu (objekt typu *USRPRF). Tento objekt identifikuje uživatele systému a definuje oprávnění a funkce, které může uživatel v systému vykonávat. v AS/400 existuje 5 uživatelských typů:

- ***SECOFR** (*Security Officer*) správce zabezpečení
- ***SECADM** (*Security Administrator*) administrátor zabezpečení
- ***PGMR** (*Programmer*) programátor
- ***SYSOPR** (*System Operator*) systémový operátor
- ***USER** (*User*) uživatel

Oprávnění k přístupu k objektům a provádění jednotlivých operací v rámci tříd uživatelů ilustruje následující tabulka:

User Class	**Special Authorities**					
*SECOFR	*ALLOBJ	*SECADM	*SAVSYS	*JOBCTL	*SERVICE	*SPLCTL
*SECADM		*SECADM	*SAVSYS	*JOBCTL		
*PGMR			*SAVSYS	*JOBCTL		
*SYSOPR			*SAVSYS	*JOBCTL		
*USER	*NONE (NO SPECIAL AUTHORITIES ARE ASSIGNED)					

Tab. č. 1: Oprávnění a typy
(Zdroj: [12])

***ALLOBJ** (*All Object*): Uživatel s tímto oprávněním má přístup ke všem objektům v AS/400 (toto oprávnění má pouze třída *SECOFR).
***SECADM** (*Security Administration*): Uživatel s tímto oprávněním může přidávat, měnit a mazat uživatele a uživatelské profily.
***SAVSYS** (*Save System*): Uživatel s tímto oprávněním může ukládat a obnovovat soubory, ke kterým je autorizován.
***JOBCTL** (*Job Control*): Uživatel s tímto oprávněním může měnit, zobrazovat, pozastavit, uvolnit, zrušit a smazat všechny úlohy v systému.
***SERVICE** (*Service*): Uživatel s tímto oprávněním může využívat nástroje systémových služeb, které umožňují sledovat data na komunikačních linkách.
***SPLCTL** (*Spool Control*): Uživatel s tímto oprávněním může zobrazovat a mazat soubory vlastněné jinými uživateli.
***NONE** (*None*): Uživatel nemá žádné speciální oprávnění.

Uživatelský profil obsahuje seznam všech objektů, které uživatel vlastní[10], a také seznam všech objektů, ke kterým má uživatel oprávnění.

Oprávnění k objektům (*object authorities*) a datům (*data authorities*) poskytují možnost pracovat s objekty i jejich datovým obsahem podle typu oprávnění. V AS/400 jsou oprávnění k objektům a datům následující:

Oprávnění k objektu

- ***OBJEXIST** Umožňuje ukládat nebo odstraňovat objet.
- ***OBJMGT** Umožňuje přemisťovat nebo přejmenovávat objekt.
- ***OBJALTER** Umožňujeí změnit atributy objektu.
- ***OBJOPR** Operační oprávnění nutné jako podmínka k oprávnění k datům.

Oprávnění k datům (je nutné mít zároveň oprávnění `*OBJOPR`)

- ***READ** Umožňuje číst obsah objektu.
- ***ADD** Umožňuje přidávat do obsahu objektu.
- ***UPDATE** Umožňuje měnit obsah objektu.
- ***DELETE** Umožňuje smazat obsah objektu.
- ***EXECUTE** Umožňuje spouštění (programů) a hledání (např. hledání knihoven).

4.3.2 Control Languague

Control Language (CL) je řídící jazyk AS/400. CL je schopný překladu a je zároveň integrovanou součástí operačního systému. *Control Language* poskytuje CL příkazy, které jsou základem interakce se systémem.

Samotný CL příkaz je objekt typu `*CMD`. Každý příkaz se skládá ze jména a pojmenovaných parametrů podle definice příkazu. Většina CL příkazů může být použita dvěma způsoby:

- zápisem samotného CL příkazu na příkazovou řádku
- jako součást dávek úloh (*jobs*)
- seskupením několika CL příkazů a vytvoření CL programu

Většinu CL příkazů dodává IBM jako součást operačního systému, lze si však vytvořit i vlastní CL příkaz. CL dodávaný IBM obsahuje sadu více než tisíce příkazů, nicméně CL je poměrně jednoduchý na naučení a pochopení, protože systémové příkazy (i když existují výjimky) vychází z následujících pravidel:

- 1. až 3. znak je zkratka anglického slovesa a vyjadřuje druh činnosti příkazu.
- 4. až 10. znak je předmět (objekt), s kterým se bude pracovat, případně další informace.

Příklad CL příkazu:

`SNDMSG` příkaz, který slouží k zasílání (*send* = SND) zprávy (*message* = MSG) od uživatele do fronty zpráv.

[10] Vlastníkem objektu je ten uživatel, který objekt vytvořil.

4.3.3 Úlohy a subsystémy

Úloha (*job*) je určitý proces (posloupnost činností). V rámci úlohy jsou spouštěny programy a CL příkazy. Každá úloha má svoje označení, svůj počátek (zahájení úlohy) a svůj konec (ukončení úlohy).

Subsystém je ohraničené prostředí, v kterém běží úlohy. Pomocí subsystémů se celý systém dělí na menší celky. Tyto subsystémy pracují na sobě nezávisle, ale sdílejí společné prostředky počítače. IBM dodává se systémem některé již vytvořené subsystémy (v knihovně `QSYS`), lze však vytvořit i subsystémy vlastní.

4.3.4 Uživatelské operace s objekty (CL příkazy, CL program)

Jak již bylo zmíněno výše, uživatel může pracovat s objekty systému pomocí samostatných CL příkazů, vytvořením programu (procedury) seskupujícím jednotlivé příkazy, nebo v rámci jednotlivých úloh, do kterých jsou příkazy, programy a procedury použity.

CL příkazy:

Na tomto místě uvedeme pro ilustraci ještě několik dalších CL příkazů spolu s jejich popisem:

Některé slovesné zkratky používané v rámci CL příkazů:

ADD	(*Add*)	přidat
CRT	(*Create*)	vytvořit
CLR	(*Clear*)	vymazat
CPY	(*Copy*)	kopírovat
CHG	(*Change*)	změnit
DLT	(*Delete*)	smazat

Příklady CL příkazů:

ADDMSGD	Přidat (*add* = ADD) definici (*definition* = D) zprávy (*message* = MSG).
CHGLIB	Změnit (*change* = CHG) atributy knihovny (*library* =LIB).
CLRMSGQ	Smazat obsah (*clear* = CLR) fronty (*queue* = Q) zpráv (*message* = MSG).
CPYDOC	Zkopírovat (*copy* = CPY) dokument (*document* = DOC) z jednoho adresáře do druhého.
CRTUSRPRF	Vytvoření (*create* = CRT) uživatelského (*user* = USR) profilu (*profile* = PRF).
DLTF	Smazat (*delete* = DLT) CL soubor (*file* = F)
DSPPGM	Zobrazit (*display* = DSP) informace o program (*program* = PGM).
WRKOBJ	Pracovat (*work* = WRK) s objektem (*object* = OBJ).

Seznam všech systémových příkazů je dostupný na www stránkách *IBM iSeries Information Center* (zdroj [32]) v sekci: *programování / Control language / Alphabetic list of commands*.

CL program:

Samotný CL program je objekt, který v sobě zahrnuje jeden či více modulů (popis modulu bude uveden dále). Program zároveň musí mít vstupní proceduru programu, která je vytvořena při kompilaci. CL kompilátor generuje vstupní programovou proceduru pro každý modul zvlášť.

Modul:

Je objekt, který je výsledkem kompilace (pokud proběhla bezchybně). Moduly však nemohou být spuštěny pokud nejsou zahrnuty do programu. Samotný modul se skládá ze dvou částí:

- z uživatelem napsané procedury
- ze vstupní programové procedury generované kompilátorem

Procedura:

Je seskupení CL příkazů. Po provedení těchto příkazů se běh vrací do místa odkud je procedura volána. Každá procedura má své vlastní jméno, kterým ji lze volat. Zdrojový text CL procedury obsahuje zdrojové řádky obsahující CL příkazy (případně i komentáře).

Strukturu CL procedury v souvislosti s možným použitím CL příkazů pro jednotlivé části procedury ilustruje následující obrázek:

Sekce procedury	CL příkazy
Začátek	**PGM (nepovinné)**
Deklarace	**COPYRIGHT, DCL, DCLF**
Globální monitor	**MONMSG**
Tělo procedury	**Prováděcí příkazy jazyka CL** SNDPGMMSG, DLTF, CHGLIB, ADDMSG... **Logické příkazy** IF, THEN, ELSE, DO, ENDDO, GOTO **Předdefinované funkce** %SUBSTRING (%SST), %SWITCH, and %BINARY (%BIN) **Příkazy pro volání programu a návrat z programu** CALL, RETURN **Příkazy pro volání procedury a návrat z procedury** CALLPRC, RETURN
Konec	**ENDPGM (nepovinné)**

Obr. č. 2: Struktura CL procedury

Začátek: CL procedura začíná příkazem `PGM`. Pokud tento příkaz není uveden, překladač jej sám doplní.

Deklarace: V této sekci se také deklarují proměnné (příkaz `DCL`) a soubory (příkaz `DCF`) (pokud jsou proměnné a soubor v proceduře použity). Může zde být také použit příkaz `COPYRIGHT`.

Globální monitor: Globální monitor chyb je spuštěn příkazem `MONMSG`. Tento monitor reaguje na každou neošetřenou chybu. V případě chyby může být v globálním monitoru uveden skok (příkazem `GOTO`) na chybové návěští. Pokud není skok na chybové návěští definován, budou všechny neošetřené chyby v programu ignorovány.

Příkazy v těle procedury: Tato sekce obsahuje samotné systémové nebo uživatelem definované řídící CL příkazy.

Konec: CL procedura končí příkazem `ENDPGM`. Pokud tento příkaz není uveden, překladač jej sám doplní.

Příklad CL procedury v AS/400 může vidět na následujícím obrázku.

```
 Columns . . . :   1  80                          Edit                                AEGMVR/QCLSRC
 SEU==>                                                                                POKUS04
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
       ********** Beginning of data **************************************************************************
0001.00            PGM       /* komentář */                                      081011
0002.00            DCL       VAR(&OBJEKT) TYPE(*CHAR) LEN(10)                    081011
0003.00            DCL       VAR(&KNIHOVNA) TYPE(*CHAR) LEN(10)                  081011
0004.00            DCL       VAR(&TYPE) TYPE(*CHAR) LEN(8)                       081011
0005.00            MONMSG    MSGID(CPF0000) EXEC(GOTO CMDLBL(CHYBA))             081011
0006.00            DSPOBJD   OBJ(&KNIHOVNA/&OBJEKT) OBJTYPE(&TYPE) +             081011
0007.00                        OUTPUT(*OUTFILE) OUTFILE(QTEMP/SEZNAMOBJ)         081011
0008.00            RETURN                                                        081011
0009.00  CHYBA:    DMPCLPGM                                                      081011
0010.00            SNDPGMMSG  MSG(POZOR) TOPGMQ(*SAME)                           081011
0011.00            SNDPGMMSG  MSGID(ACL0001) MSGF(ACLMSG) MSGDTA('ACL01C') +     081011
0012.00                         MSGTYPE(*ESCAPE)                                 081011
0013.00       MONMSG  MSGID(CPF0000)                                             081011
0014.00            ENDPGM                                                        081011
       ********** End of data ****************************************************************************
```

Obr. č. 3: CL procedura v AS/400

Příklad ukazuje proceduru, kde jsou deklarovány (`DCL`) proměnné `&OBJEKT`, `&KNIHOVNA` a `&TYPE`. Globální monitor pak monitoruje chybu při provádění příkazů v těle procedury – provádění příkazu `DSPOBJD` (*display objecjt definition*), který zobrazuje atributy konkrétního objektu (`&OBJEKT`) v příslušné knihovně (`&KNIHOVNA`). Pokud při provádění příkazu nastala chyba, globální monitor ji zaregistruje a v proceduře provede skok na definované návěští (`CMDLBL`), kde vytvoří 2 programové zprávy. Jednu okamžitou zprávu s textem „`POZOR`“ a jednu předdefinovanou únikovou zprávu (`MSGID ACL001`), která vede k ukončení programu. Zprávami v AS/400 se následně budeme zabývat v následující kapitole.

4.4 Systém zpráv na AS/400

4.4.1 Zprávy

Zpráva je krátké textové sdělení, prostřednictvím kterého spolu navzájem komunikují:

- programy a uživatelé systému
- programy navzájem
- procedury uvnitř programu
- uživatelé systému navzájem

Zprávy jsou rozděleny na **okamžité** (*immediate*) a **předdefinované** (*predefined*):

- **Okamžité zprávy** jsou vytvořeny a posílány programem nebo uživatelem systému interaktivně. Nemají identifikaci a nejsou permanentně uloženy v systému. Příklad zobrazení takové zprávy může vypadat následovně:

```
From . . . : QSYSOPR 06/12/94 10:50:54
System going down at 11:00; please sign off
```

- **Předdefinované zprávy** jsou již vytvořeny před tím, než jsou použity. Lze je poslat pouze z programu. Tyto zprávy mají vlastní identifikaci (ID) a jsou uloženy v systému permanentně v souboru zpráv (*message file*). Pokud mají být použity, jsou z tohoto souboru na základě své identifikace vyzvednuty. Předdefinovaná zpráva může při výpisu na obrazovku vypadat například následujícím způsobem:

```
CPF0006 Errors occurred in command.
```

V AS/400 existuje několik typů zpráv:

- **`*INFO` (*Information*)** Informační zpráva, která nevyžaduje odpověď. Může být momentální i předdefinovaná. Obsahuje text pouze informativního charakteru. Může být součástí programu, nebo pomocí těchto zpráv mohou například mezi sebou interaktivně komunikovat jednotliví uživatelé systému.
- **`*INQ` (*Inquiry*)** Dotazová zpráva. Stejně jako informativní zpráva může být předdefinovaná nebo momentální zasílaná z programů i mimo ně uživateli systému, avšak vyžaduje odpověď.
- **`*RPY` (*Reply*)** Odpověď na dotazovou zprávu. Může být okamžitá, nebo přednastavená v definici předdefinované zprávy.
- **`*DIAG` (*Diagnostic*)** Diagnostická zpráva. Je zasílána z programů a informuje o chybách detekovaných v proceduře nebo programu.
- **`*NOTIFY` (*Notify*)** Upozornění na nějakou konkrétní skutečnost. Může být zasílána pouze z programu.
- **`*ESCAPE` (*Escape*)** Úniková zpráva zasílaná z programu, která informuje o tom, že program nebo procedura byly ukončeny abnormálním způsobem.
- **`*COMP` (*Completion*)** Informuje o dokončení nějakého procesu. Například o ukončení úlohy. Může být zasílána pouze z programu.
- **`*STATUS` (*Status*)** Zpráva, která informuje o stavu (statutu) programu. Může být zasílána pouze z programu.
- **`*RQS` (*Request*)** Požadavek na operační systém. Samotné CL příkazy jsou zprávy typu `*RQS`.

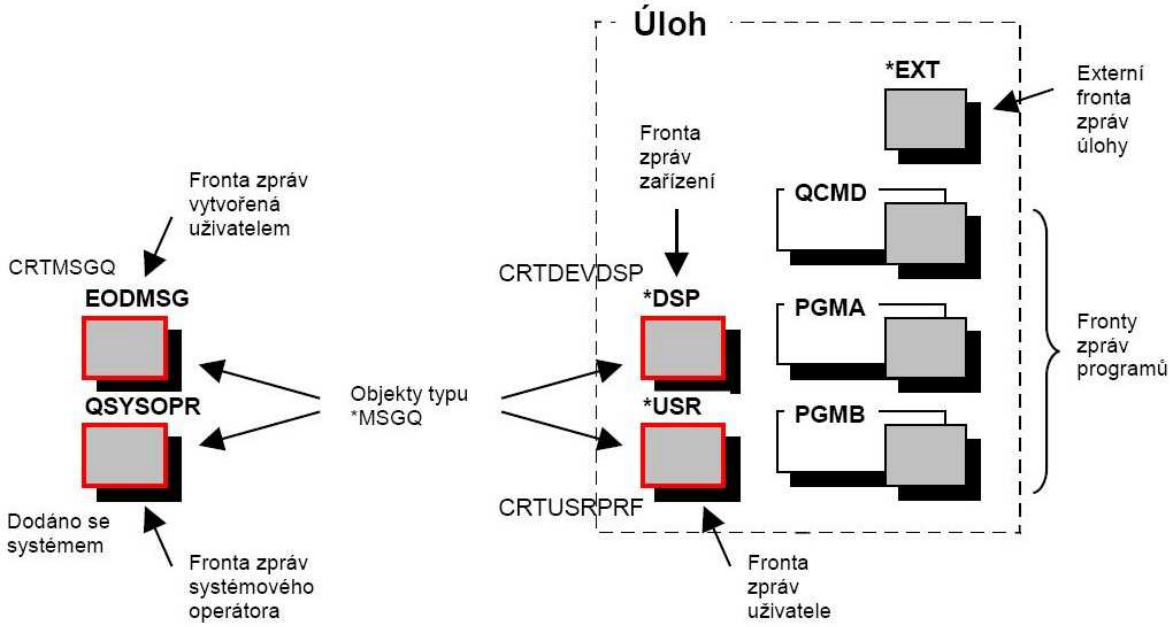
4.4.2 Fronty zpráv

Zprávy jsou zasílány do příslušných front zpráv, které jsou samy objekty systému a jsou přiřazeny jednotlivým programům, úlohám a uživatelům. Tyto fronty zpráv jsou vytvářeny systémem, nebo pomocí příkazů pro tvorbu front uživatelem.

Pokud je například vytvořen popis nového zařízení (objekt typu `*DSP`), pak je vytvořena uživatelem nebo pak automaticky systémem fronta zpráv zařízení se stejným jménem, jako je jméno zařízení.

Při vytváření uživatelského profilu je vytvořena fronta zpráv v knihovně `QUSRSYS` podle jména uživatelského profilu. Pro uživatelský profil lze použít i již vytvořenou frontu zpráv, a to buď vytvořenou pomocí `CRTMSGQ` nebo systémovou – některé fronty jsou dodány již se systémem (např. fronta systémového operátora `QSYSOPR`).

Existují také fronty zpráv jednotlivých úloh. Jedná se o dočasnou konstrukci pro běh úlohy vytvářenou při inicializaci úlohy. Externí část (`*EXT`) fronty úlohy je používána programy pro komunikaci (pomocí zpráv) s uživatelem. Pokud je v rámci úlohy inicializován program, je přidána do fronty zpráv úlohy fronta zpráv tohoto programu. Fronty zpráv programů jsou používány na komunikaci mezi programy v zásobníku programů úlohy. Fronty zpráv v systému AS/400 ilustruje následující obrázek.



Obr. č. 4: Fronty zpráv AS/400

(Zdroj: [4])

4.4.3 Příkazy pro zasílání zpráv

Jak již bylo řečeno, pomocí zasílání zpráv spolu vzájemně komunikují programy (procedury) a uživatelé systému, přičemž nezáleží na tom, na jaké pracovní stanici běží příslušný program nebo je uživatel přihlášen.

Zprávy jsou zasílány do příslušných front zpráv, přičemž jednu zprávu lze zároveň zaslat naráz do více front zpráv (tedy i více uživatelům najednou). AS/400 poskytuje pro zasílání okamžitých i předdefinovaných zpráv několik CL příkazů:

Příkazy pro zasílání výhradně okamžitých zpráv:

- **SNDMSG** (*send message*) Příkaz umožňuje uživateli zaslat okamžitou zprávu do jedné nebo více front zpráv. Text je zadán do parametru MSG, přičemž je nutné zadat příjemce zprávy (frontu zpráv příjemce). Zpráva může být také typu *INQ, tedy zpráva vyžadující odpověď. Příklad:

```
SNDMSG   MSG ('How long will the display stations be online today?')
TOMSGQ(*SYSOPR)  MSGTYPE(*INQ)
```

  Příklad ukazuje zaslání zprávy od uživatele do fronty zpráv systémového operátora (*SYSOPR), přičemž zpráva je typu *INQ a vyžaduje odpověď.

- **SNDBRKMSG** (*send break message*) Příkaz používá systémový operátor (*SYSOPR) k zasílání okamžitých zpráv do front jedné, či více pracovních stanic. Tato zpráva způsobí přerušení činnosti tzv. *break mode*, který pozastaví činnost uživatele, dokud nepotvrdí její přečtení. Příklad:

```
SNDBRKMSG   MSG('Your printed output is ready.')
TOMSGQ(GEORGEMSGQ)
```

  V tomto příkladu systémový operátor zasílá přerušovací zprávu do konkrétní fronty GEORGEMSGQ.

- **SNDNETMSG** (*send network*) Tento příkaz umožňuje zasílat okamžitou zprávu typu *INFO prostřednictvím sítě uživateli na vzdálené stanici. Příklad:

```
SNDNETMSG   MSG('Please do not make any more changes to the
accounts receivable files for the next hour.') TOUSRID((SMITH
SYSTEM2))
```

  Příkaz zašle zprávu (text zprávy v parametru MSG) uživateli identifikovaným jménem jako SMITH, na pracovaní stanici se jménem SYSTEM2.

Příkazy pro zasílání okamžitých i předdefinovaných zpráv:

- **SNDPGMMSG** (*send program message*) Pomocí tohoto příkazu lze poslat okamžitou i předdefinovanou zprávu, na kterou však není vyžadována odpověď. Příkaz SNDPGMMSG je také možné použít pouze v programu (proceduře). U okamžité zprávy stačí zadat pouze text do parametru PGM, typ zprávy a kam (do jaké fronty) se má zpráva poslat. U předdefinované zprávy je nutné zadat MSGID a soubor zpráv, kde je zpráva uložena. U obou případů je pak možné zadat i další parametry. Příklad:

```
SNDPGMMSG   MSGID(UIN0023)  MSGF(INV) MSGDTA('50 100') TOPGMQ(*EXT)
```

  V tomto příkladě je do externí fronty zpráv úlohy (*EXT) zaslána zpráva s MSGID(UIN0023), uložená v souboru zpráv INV, přičemž v parametru MSGDTA jsou uloženy hodnoty, které jsou následně dosazeny do příslušných proměnných předdefinované zprávy.

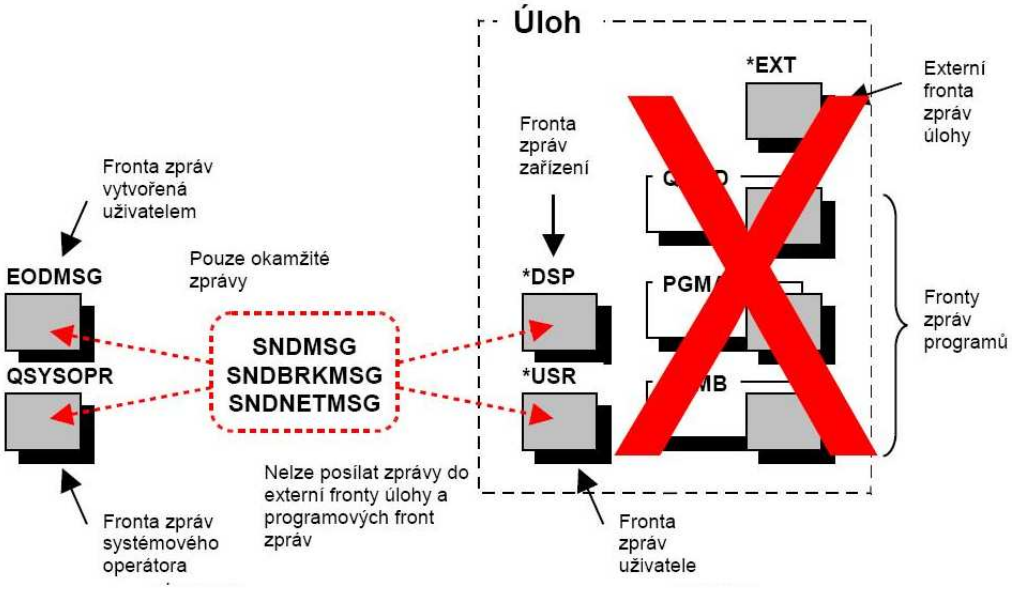
- **SNDUSRMSG** (*send user message*) Tento příkaz může být použit pouze z programu a lze pomocí něho poslat okamžitou i předdefinovanou zprávu, na kterou je zpravidla vyžadována odpověď. Příklad:

```
SNDUSRMSG MSG('Data has been verified. Do you want to update the
master files (Y or N)?') TOMSGQ(*) VALUES(Y N) DFT(N) MSGRPY(&REPLY)
```

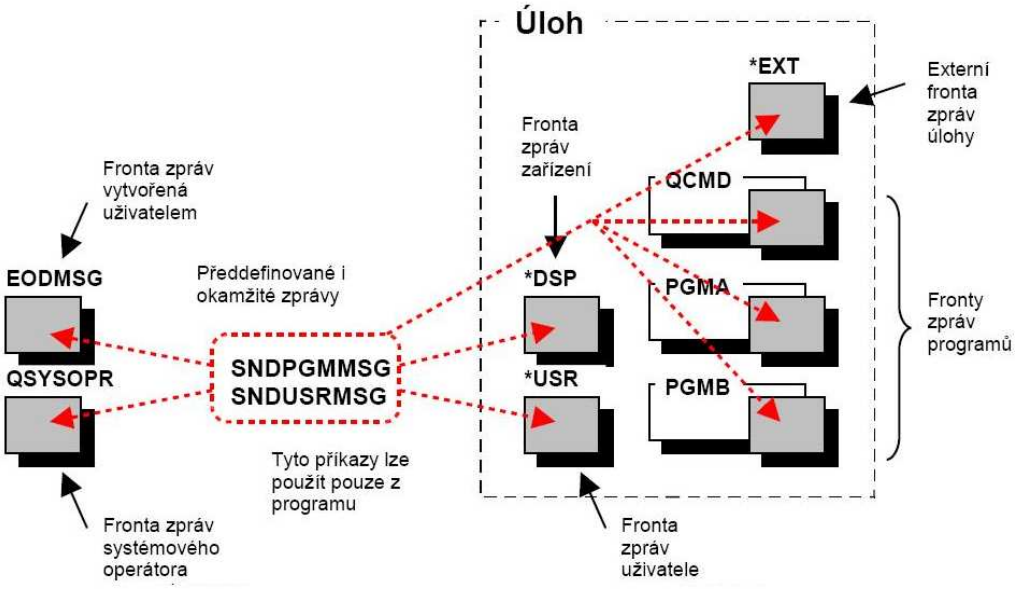
V příkladě je uživateli pracovní stanice zaslána dotazová zpráva (typ `*INQ`), zda chce provést *update* souborů, přičemž se čeká na jeho volbu `Y/N`. Volba `Y` nebo `N` je pak zaslána v odpovědi v proměnné `&REPLY`.

Příkazy `SNDMSG`, `SNDBRKMSG` a `SNDNETMSG` tedy mohou být zadány jak interaktivně uživatelem z příkazové řádky, tak jako součást programu. Těmito příkazy lze zasílat zprávy pouze do front, které jsou v systému perzistentním objektem. Nelze je tedy zasílat do front zpráv úlohy. Viz obrázek č. 5.

Příkazy `SNDPGMMSG` a `SNDUSRMSG` lze použít pouze z programu a lze jimi posílat okamžité i předdefinované zprávy do všech front zpráv (tedy i do front zpráv úloh). Viz obrázek č. 6.



Obr. č. 5: Fronty a příkazy SNDMSG, SNDBRKMSG, SNFNETMSG
(Zdroj: [4])

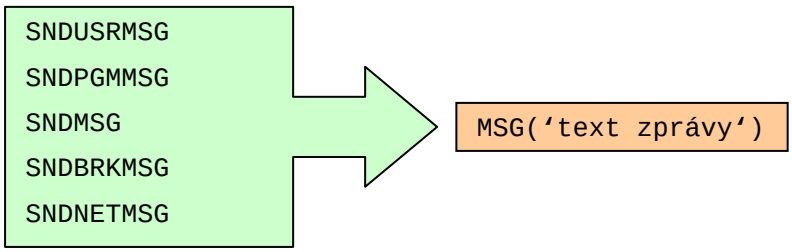


Obr. č. 6: Fronty a příkazy SNDPGMMSG, SNDUSRMSG

(Zdroj: [4])

Text zprávy – okamžité zprávy:

Text zprávy je u okamžitých zpráv vložen do parametru `MSG()` v CL příkazu pro zaslání zprávy.



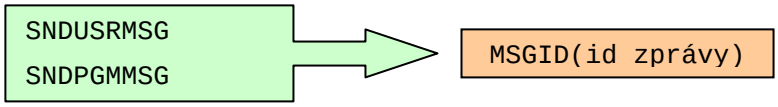
Příklad:

```
SNDBRKMSG MSG('System going down in 15 minutes') TOMSGQ(*ALLWS)
```

V tomto příkladě zasílá systémový operátor do front všech stanic (`ALLWS` - *all work stations*) okamžitou zprávu, že za 15 minut bude vypnut systém, přičemž se jedná o zprávu přerušovací, tedy takovou, která pozastaví činnost uživatele dokud nepotvrdí její přijetí.

Text zprávy – předdefinované zprávy:

U předdefinovaných zpráv je text uložen v definici zprávy. Není tedy součástí příkazu pro zasílání zprávy. Text zprávy je zobrazen až při výpisu zprávy na výstup (obrazovka, soubor). Konkrétní zpráva v souboru zpráv je pak identifikována v parametru `MSGID()`.



Příklad:

```
SNDPGMMSG MSGID(USR4310) MSGF(QGPL/USRMSG) + MSGDTA(&CUSNO) TOPGMQ(*EXT) +
MSGTYPE(*INFO)
```

V příkladu je z programu zaslána do externí fronty zpráv úlohy informační zpráva s ID: `USR4310`, která je uložená v souboru `USRMSG` v knihovně `QGPL`. Má již předdefinovaný text: „`Customer number &1 not found`“, který také obsahuje proměnnou `CUSNO`. Výsledná zpráva pak bude po dosazení hodnoty do proměnné `&CUSNO` vypadat při zobrazení na obrazovce následovně:

```
Customer number 35500 not found
```

Podrobněji se předdefinovanými zprávami budeme zabývat dále.

4.4.4 Předdefinované zprávy

Předdefinované zprávy jsou uloženy **v souborech zpráv** (objekty typu `*MSGF`) a lze je používat opakovaně. Zasílání předdefinovaných zpráv je však možné **pouze z programů**. Do **textů** těchto zpráv mohou být zakomponovány **proměnné**, které jsou naplňovány při zasílání zprávy. Každá zpráva je identifikována svým ID, které je dlouhé sedm znaků (tři znaky a čtyři číslice, např. `CHK0002`). **Zprávy** pak **nejsou** uloženy **ve frontě zpráv celé**, ale jsou uložena **pouze jejich ID a data** v podobě hodnot proměnných.

Předdefinované zprávy mohou být systémové (dodávané se systémem jako jeho součást – uložené v souboru `QCPFMSG` v knihovně `QSYS`)[11], nebo vytvořené uživatelem.

Vytvoření vlastního souboru zpráv:

Pokud chce uživatel vytvořit vlastní předdefinované zprávy, musí nejprve vytvořit vlastní soubor zpráv. K tomu slouží příkaz **`CRTMSGF`**, který má následující parametry:

```
MSGF( ) SIZE( ) AUT( ) TEXT( )
```

První parametr, do kterého je vkládán název souboru je povinný, ostatní parametry: velikost, oprávnění k přístupu k souboru a textový popis jsou nepovinné. Příkaz pro vytvoření souboru zpráv může například vypadat následovně:

```
CRTMSGF MSGF(QGPL/USRMSG) + TEXT('Message file for user-
created messages')
```

Tento příkaz vytvoří soubor zpráv se jménem `USRMSG`, v knihovně `QGPL`, s textovým popisem, že se jedná o soubor zpráv pro uživatelem vytvářené zprávy.

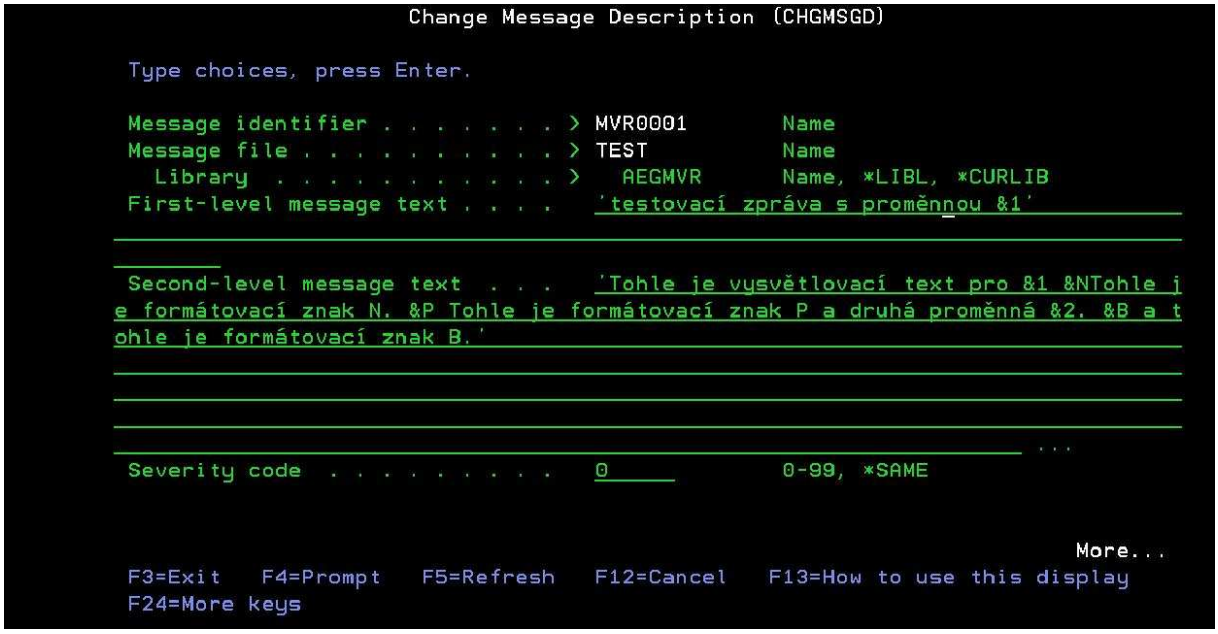
Přidání vlastních zpráv do souboru zpráv:

Pokud má uživatel vytvořen vlastní soubor zpráv, může v něm vytvářet předdefinované zprávy pomocí příkazu **`ADDMSGD`** (*add message description*). Při vytváření definice nové zprávy (`ADDMSGD`) je zadáváno ID zprávy, jméno souboru zpráv, ve kterém bude zpráva uložena a dále popis zprávy. Důležité atributy popisu zprávy jsou především:

[11] Licenscované aplikace často používají vlastní soubory zpráv. Například program *COBOL* používá fronty zpráv QCBLMSG, QCSCMSG a QLNCMSG, program *RPG* používá soubor zpráv QRPGMSG a QRPGLEMSG a podobně.

- Text zprávy (povinný parametr) – text první úrovně. Je základním popisem obsahu zprávy, přičemž umožňuje použití proměnných v textu zprávy.
- „Help text“[12] – text druhé úrovně. Je doplňujícím popisem zprávy. Popisuje rozšiřující informace nebo konkrétní možnosti užití zprávy. Text druhé úrovně taktéž umožňuje použití proměnných.
- Kód závažnosti zprávy (*severity code*) je číselný kód v rozmezí 10 až 99, určující důležitost zprávy.
- Popis formátu dat zprávy v parametru `FMT`. Zde jsou zadávány typ[13] a velikost jednotlivých proměnných zprávy.

Následující obrázek ukazuje obrazovku při zadání příkazu `CHGMSGD` a ilustruje změnu definice zprávy v AS/400.



Obr. č. 7: Obrazovka AS/400 při zadání příkazu CHGMSGD – změna definice zprávy

Na obrázku můžeme vidět obrazovku po zadání příkazu pro změnu definice zprávy (`CHGMSGD`) u zprávy s identifikátorem `MVR0001`, uloženém v souboru `TEST` v knihovně `AEGMVR`. Pod identifikačními údaji zprávy jsou řádky pro text zprávy (text první úrovně), který v tomto případě obsahuje text s informací, že se jedná o testovací zprávu s proměnnou `&1` a doplňující text druhé úrovně (v našem případě obsahuje také formátovací znaky pro zobrazování textu – `&N`, `&P`, `&B`, které zajišťují odsazení nového řádku o různé hodnoty – formátovacími znaky se však nebudeme podrobněji zabývat). Dole pak vidíme možnost definování závažnosti (*severity code*). Další parametry (např. `FMT`) lze zobrazit při zvolení možnosti `More`.... Některými důležitými parametry v rámci popisu zprávy se budeme zabývat dále.

[12] AS/400 poskytuje kontextový help při stisku klácesy F1.
[13] Například: *BIN (binární), *DEC (decimální), *ITV (časový interval), *DTS (časové razítko). Výčet všech datových typů, pro proměnné zpráv lze nalézt v [13] *Chapter 7. Defining Messages*.

Závažnost zprávy:

Závažnost zprávy (*severity code*) představuje rozdělení zpráv na několik úrovní. Kódové označení závažnosti popisuje dvoumístná číslice, přičemž platí, že čím vyšší číslo, tím vyšší závažnost zprávy. Jednotlivé typy závažnosti zpráv jsou následující:

- **00: Information:** Zpráva tohoto typu má za cíl pouze informovat. Neznamená žádnou chybu nebo vyžadovanou odpověď.
- **10: Warning:** Varování před potencionálním výskytem chyby.
- **20: Error:** Zpráva oznamuje výskyt chyby.
- **30: Severity error:** Zpráva informuje o výskytu chyby považované za závažnou.
- **40: Abnormal end of procedure or function:** Zpráva tohoto typu informuje o nestandardním ukončení procedury nebo funkce (přerušená uživatelem nebo kvůli chybným datům v proceduře).
- **50: Abnormal end of job:** Zpráva tohoto typu informuje o nestandardním ukončení úlohy (úloha byla nestandardně ukončena nebo nemohla vůbec začít).
- **60: System status:** Informuje o stavu a případných varováních v rámci systému, subsystémů a zařízení. Určená pouze pro systémového operátora.
- **70: Device integrity:** Indikuje, že určité zařízení nefunguje nebo není schopné provozu. Určená pouze pro systémového operátora.
- **80: System alert:** Varuje, že systém by nemusel být provozuschopný, pokud nebudou učiněna potřebná opatření.
- **90: System integrity:** Indikuje, že systém nebo subsystém není provozuschopný. Určená pouze pro systémového operátora.
- **99: Action:** Upozorňuje, že je třeba učinit nějakou konkrétní akci (např. napsat odpověď na zprávu).

Proměnné v textu zprávy:

Proměnná v textu zprávy je uvozena znakem `&` a za ní následuje číslo od 1 do 99. Proměnné zprávy musí být číslovány lineárně a vzestupně (`&1`, `&2`, `3&`…), přičemž jednotlivá čísla nelze přeskakovat. Proměnné ve zprávě poskytují nástroj k tomu, jak učinit zprávy více konkrétními. Mohou obsahovat identifikaci konkrétního objektu, kterého se zpráva týká. Například zpráva typu:

```
File not found
```

dostává s přiřazením jména souboru konkrétnější význam, pokud je ve zprávě sděleno i jméno souboru, který nebyl nalezen:

```
File ORDHDRP not found
```

Jméno souboru můžeme ve zprávě tedy definovat jako proměnnou, která bude obsahovat identifikaci nenalezeného souboru:

```
File &1 not found
```

v parametru `FMT` bude tato proměnná definována jako řetězec (např. velikosti 10 znaků):

```
FMT((*CHAR 10))
```

Text zprávy může obsahovat i více proměnných například:

```
Object &1 of type &3 in library &2 is not available
```

Potom musí být také současně uvedeny parametry FMT například:
```
FMT((*CHAR 10) (*CHAR 10) (*CHAR 7))
```

Příklad použití popisu zprávy s proměnnou:

```
ADDMSGD MSGID(USR4310) + MSGF(QGPL/USRMSG) +
MSG('Customer number &1 not found') +
SECLVL('Change customer number') +
SEV(40) + FMT((*CHAR 8))
```

Příklad ukazuje změnu definice zprávy s ID USR4310, uložené v souboru USRMSG, v knihovně QGPL. Zpráva obsahuje text první úrovně, který informuje o tom, že číslo zákazníka nebylo nalezeno. Číslo zákazníka je vkládáno do zprávy jako proměnná &1, jejíž délka a typ je definovaná v parametru FMT (v tomto případě jako řetězec o délce maximálně osmi znaků). Zpráva také obsahuje text druhé úrovně, vyzývající uživatele, aby změnil číslo zákazníka. Zpráva má definovanou závažnost 40 (abnormální konec procedury nebo funkce).

Příklad zaslání zprávy s konkrétní hodnotou proměnné:

Pokud chceme následně poslat tuto zprávu do nějaké konkrétní fronty zpráv, bude použito parametru s identifikací zprávy, název souboru zpráv, kde je tato zpráva uložena, cílová destinace (název fronty kam je zpráva zasílána) a parametr **MSGDTA s hodnotou**, která bude následně dosazena do proměnné zprávy &1:

```
SNDPGMMSG  MSGID(USR4310) MSGF(USRMSG) MSGDTA('10086') TOPGMQ(*EXT)
```

Zobrazení zprávy:

Při zobrazení zprávy na obrazovku bude zpráva „poskládáná“ z konkrétního předdefinovaného textu a hodnot proměnných, které budou do textu namísto proměnných dosazeny.

```
USR4310 Customer number 10086 not found
```

4.4.5 Logování zpráv v operačním systému AS/400

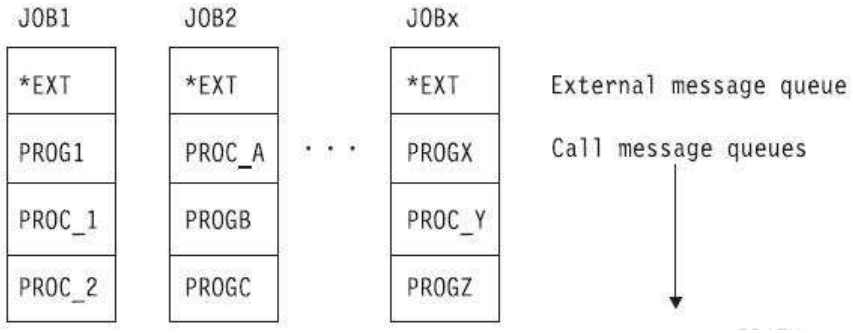
Zasílané zprávy jsou zároveň ukládány do logu, který je součástí systému. V AS/400 Existují dva typy logů pro ukládání zpráv:

1) **log úlohy (*job log*)**
2) **log historie (*history log*)**

Log úlohy

Každá úloha má svůj vlastní log úlohy v podobě fronty zpráv úlohy, která však existuje pouze po dobu trvání úlohy. Pokud chceme tento log uchovat, může být po skončení úlohy obsah logu úlohy zapsán do výstupního souboru (QPJOBLOG) nebo do databáze.

Fronta zpráv úlohy se sestává z externí fronty zpráv úlohy (*EXT) a zásobníku, který obsahuje odkaz na jednotlivé fronty programů a procedur volaných v úloze. Viz následující obrázek.



Obr. č. 8: Fronty zpráv úloh
(Zdroj: [13])

Do logu úlohy jsou ukládány příkazy provedené v rámci úlohy (CL příkazy jsou zprávy typu *RQS), všechny zprávy zaslané v subsystému v průběhu úlohy a je také možné do logu ukládat i příkazy z jednotlivých procedur. Pomocí nastavení parametru LOG v příkazech CRTCLMOD (*Create Control Language Module*) nebo CRTBNDCL (*Create Bound Control Language Program*), které slouží k vytvoření modulu (respektive programu) z dané procedury, můžeme specifikovat, zda budou příkazy v CL proceduře ukládány do logu úlohy (*job log*) – respektive do fronty zpráv úlohy. Pokud je tedy parametry logování nastaven na provádění logování, jsou logovací zprávy zasílány do fronty zpráv procedury, které jsou pak součástí fronty zpráv úlohy. Log pak obsahuje čas, jméno procedury a programu, text zprávy, jméno příkazu s parametry a jejich obsah. Pomocí příkazu RMVMSG lze pak jednotlivé příkazy z fronty zpráv (z logu) odstranit.

Lze také provádět filtraci zpráv a určovat jaké informace (zprávy) má systém do logu úlohy zapisovat. A to na základě nastavení LOG parametrů v příkazu CHGJOB (*Change Job*), které umožňují filtrovat zprávy a zapisovat do logu úlohy zprávy pouze určité úrovně závažnosti s možností zapisování, či nezapisování textu první a druhé úrovně.

Zobrazovat log aktivní úlohy je možné pomocí příkazu DSPJOBLOG.
Například:

```
DSPJOBLOG JOB(nnnnnn/AEGMVR/WS1)
```

zobrazí log úlohy (úloha musí být aktivní – uživatel musí být aktuálně přihlášen) uživatele AEGMVR na pracovní stanici WS1, přičemž nnnnnn je číslo úlohy.

Log historie

Log historie (QHST) zachycuje aktivity systému, informace o jednotlivých úlohách, informace o stavu zařízení a zprávy systémového operátora (*SYSOPR), který je tam může zasílat pomocí příkazu SNDMSG. Příklad:

```
SNDMSG   MSG('Input errors on PAYROLL cost
me 1 hour of run time.')  TOMSGQ(QHST)
```

Log historie se sestává z fronty zpráv a souboru nazývaného *log-version*, do kterého je log historie ukládán. Pokud je tento soubor plný, je automaticky vytvořen soubor nový. Soubory logu historie lze také pomocí příkazu `DSPOBJD OBJ(QSYS/QHST*)OBJTYPE(*FILE)` prohlížet, přičemž je možné nastavit časovou periodu, sledovat pouze určitou úlohu, nebo sledovat pouze konkrétní zprávy zadáním jejich identifikátorů. Jednotlivé záznamy v logu historie lze také odstraňovat (za pomoci příkazu `WRKOBJ OBJ(QSYS/QHST*) OBJTYPE(*FILE)` a následné volby č. 4 v menu).

Každý záznam v logu historie má následující formát:

- systémový datum a čas
- číslo záznamu
- data, která obsahují: jméno úlohy, datum a čas konvertovaný do formátu `cyymmddhhmmss`[14], ID zprávy, jméno souboru zprávy, jméno knihovny, typ zprávy[15], závažnost zprávy, jméno programu, z kterého byla zpráva odeslána[16], číslo instrukce reprezentující odesílající program, jméno programu, který zprávu přijímá[17], instrukce reprezentující přijímající program, délku textu zprávy, velikost dat proměnných, `CCSID`[18]. Tyto položky, spolu s jejich velikostí a umístěním v záznamu logu, reprezentuje následující tabulka.

Contents	Type	Length	Positions in Record
Job name	Character	26	11-36
Converted date and time[1]	Character	13	37-49
Message ID	Character	7	50-56
Message file name	Character	10	57-66
Library name	Character	10	67-76
Message type[2]	Character	2	77-78
Severity code	Character	2	79-80
Sending program name[3]	Character	12	81-92
Sending program instruction number[4]	Character	4	93-96
Receiving program name[3]	Character	10	97-106
Receiving program instruction number[4]	Character	4	107-110
Message text length	Binary	2	111-112
Message data length	Binary	2	113-114
Coded character set identifier (CCSID) for text or data[5]	Binary	4	115-118
Reserved	Character	24	119-142

Tab. č. 2: Data v záznamu logu historie

(Zdroj: [13])

[14] `c` – století (c=0 když yy ≥ 40, c = 1 když yy < 40); `yymmdd` – rok, měsíc a den kdy byla zpráva poslána; `hhmmss` – hodina, minuta a vteřina zaslání zprávy.

[15] Kód reprezentující typ zprávy. Typy zpráv viz 4.4.1 Zprávy.

[16] Musí se jednat o zkompilovaný program, nikoliv pouze o proceduru nebo modul.

[17] Zde se také musí jednat o zkompilovaný program, nikoliv pouze o proceduru nebo modul.

[18] `CCSID` (*Coded character set identifier*) určuje kódovou množinu znaků používanou ve zprávě.

4.4.6 Shrnutí důležitých vlastností systému zpráv na AS/400

- **Zprávy** mohou být buď **předdefinované** (permanentně uložené v systému), nebo **okamžité** (nejsou uloženy v systému a jejich text je vytvářen uživatelem na příkazové řádce nebo v programu interaktivně).
- Zprávy jsou vždy **zasílány/přijímány do/z příslušné fronty zpráv** (*message queue*), které jsou vždy asociovány s konkrétním programem, procedurou, nebo uživatelem. Zprávy zůstávají ve frontách zpráv, dokud nejsou příslušným programem nebo uživatelem přijaty.
- **Předdefinované zprávy** jsou definovány ve jmenném prostoru v podobě **souborů zpráv** (*message files*), tedy mimo programy, které je používají. Mají svoji **identifikaci**, **typ zprávy**, **závažnost** a další parametry.
- **Předdefinované zprávy** mají **standardizovaný text**, jehož **součástí** mohou být také **proměnné**, které lze následně zpracovávat a vyhodnocovat.
- **Při zasílání předdefinovaných** zpráv je v příkazu pro zasílání zprávy v parametru `MSGID` **zadáno ID** předdefinované zprávy, **cílová destinace** (fronta zprávy) v parametru `MSGF()` a **konkrétní hodnoty**, které se mají doplnit na místo proměnných v textu předdefinované zprávy jsou uvedeny v parametru `MSGDTA()`.
- Lze vytvářet **vlastní fronty zpráv, soubory zpráv a také definovat vlastní zprávy**.
- **Zprávy zasílané v systému** AS/400 jsou ukládány do **logu úlohy a logu historie**.

Tyto principy lze obecně uplatnit i při vývoji jiných aplikací a budou pro nás inspirací při návrhu vlastního logovacího systému. Podrobněji se jednotlivými aspekty budeme zabývat dále, přičemž možné využití některých principů systému zpráv AS/400 předložíme v kapitole *7 Definice vlastností obecného logovacího systému za použití principů AS/400*.

5 Analýza dalších vybraných logovacích systémů

5.1 Syslog a syslog-ng

Systém logování zpráv je zpravidla používán i v ostatních operačních systémech. V této části se budeme v krátkosti zabývat systémem logování syslog (respektive logovacím démonem syslogd), používaného na UNIXových systémech a z něho vzešlého logovacího systému syslog-ng (ve verzi 3.0.). Zaměříme se na to, jaké typy zpráv, úrovně závažnosti zpráv, formáty zpráv, způsoby jejich filtrování, konfigurace zdrojů a cílů a možnosti uložení logu se používají v existujících logovacích systémech. Pro upřesnění dodejme, že jako zdroj informací byly použity materiály [5], [7], [10], [11], [22], [27], [28].

5.1.1 Syslog

Obecná charakteristika:

Syslog je program běžící na pozadí jako služba syslogd (syslog démon), který v UNIXových systémech zajišťuje sběr, filtrování a ukládání systémových zpráv a také zpráv aplikací, ve kterých je tato služba volána. Původní syslog vznikl v roce 1980 jako součást projektu Sendmail[19] a postupně si unixovém světě vybudoval pozici nepsaného standardu.

Způsob fungování:

Jednotlivé zprávy jsou systémem nebo aplikacemi zasílány do socketu `/dev/log` odkud je syslogd vyzvedává a následně je zaznamenává do logovacích souborů, které lze roztřídit například podle data, závažnosti nebo zdroje, který je vytvořil, přičemž jednu zprávu lze zapsat i do více souborů najednou. Zprávy lze, abychom ukládali jenom ty pro nás důležité, před zápisem do souborů filtrovat. Způsob a kritéria filtrování zpráv jsou nastaveny v souboru `/etc/syslog.conf`, kde je lze také podle potřeby změnit. Tato kritéria jsou: závažnost (*severity*) a kategorie (*facility*), které jsou jako parametry součástí každé zprávy.

Zprávy lze také pomocí sítě ukládat na vzdálený počítač (respektive logovací server). Syslog komunikuje přes síť pomocí protokolu UDP (*User Datagram Protocol*) na portu 514. Pokud zasíláme zprávy na vzdálený počítač, je třeba uvést jeho jméno v konfiguračním souboru. Na vzdáleném počítači pak spustit syslogd s parametrem `-r`.

Pro zobrazování zpráv syslogu uložených v souborech je možné použít systémového příkazu `cat` nebo `tail`. Při zadání parametru `-f` v příkazu `tail` je možné sledovat log v reálném čase (např. `tail -f /var/log/message`)

[19] *Sendmail* je *open source software* implementující agenta pro přenos emailů.

Zprávy:

Všechny zprávy mají definovanou kategorii a závažnost. Kategorie zpráv jsou následující:

- **`auth`:** autentizace (např. zprávy týkající se přihlašování / odhlašování uživatelů)
- **`authpriv`:** autentizace, ale zprávy určeny z bezpečnostních důvodů pouze administrátorovi
- **`cron`:** zprávy od *cronu*[20]
- **`daemon`:** nespecifikované zprávy systémových aplikací
- **`kern`:** zprávy od jádra operačního systému
- **`lpr`:** zprávy z tiskového systému
- **`mail`:** logy poštovního systému
- **`mark`:** časové značky
- **`news`:** zprávy NNTP[21] (*Network News Transfer Protocol*) serveru
- **`security`:** stejné jako `auth`
- **`syslog`:** zprávy syslogu
- **`user`:** nespecifikované zprávy z uživatelských aplikací
- **`uucp`:** zprávy aplikací UUCP[22] (*Unix to Unix Copy Protocol*)

Dále existují ještě kategorie `Local0` až `Local7`, které lze použít libovolně

Syslog definuje tyto úrovně závažnosti zpráv (od nejméně závažné po nejzávažnější):

- **`debug`:** ladící hlášení
- **`info`:** informační zpráva
- **`notice`:** normální stav, ale důležitá zpráva
- **`warning`:** varovné upozornění
- **`err`:** chybové hlášení
- **`crit`:** kritický stav
- **`alert`:** výstraha – očekává se bezodkladná reakce
- **`emerg`:** systém se nachází v nepoužitelném stavu

Konfigurace:

Filtrování pomocí nastavení konfiguračního souboru má následující syntaxi:

```
{kategorie.závažnost  cílová_destinace}
```

Za kategorii i závažnost lze použít zástupný znak „`*`“ pokud chceme vybrat zprávy všech kategorií i závažností. Pokud uvedeme závažnost, je vybrána zpráva s rovnou nebo vyšší úrovní závažnosti. Pokud chceme pouze zprávy dané závažnosti, uvedeme operátor „`=`“ (například: `kern.=alert`). Pokud chceme všechny zprávy kromě dané úrovně, použijeme „`!=`“. Také je možné nastavit u jedné kategorie rozsah úrovní zpráv, které chceme logovat: `mail.info;mail.!alert`, i několik kategorií naráz stejných nebo různých úrovní závažnosti: `daemon.warning;mail.info`.

[20] *Cron* je démon, který v operačních systémech na bázi UNIXu automaticky spouští po určitém čase opakující se procesy (např. dávkové úlohy).
[21] Přenosový protokol pro síťové diskuzní skupiny na Internetu.
[22] Protokol pro přenos souborů mezi UNIXovými počítači.

Cílová destinace pak definuje cestu souboru (např.: `/var/log/auth.log`), cestu k pojmenované rouře[23] (musí být uvozena znakem „|“), terminálu, nebo jméno vzdáleného počítače (jméno musí začínat znakem „@“), kam jsou logovací zprávy ukládány.

Soubory:

Formát zprávy, která je následně zapisována do logovacího souboru je následující:

```
měsíc den čas počítač název_procesu[pid]: vlastní zpráva
```

Logy lze podle konfigurace cílové destinace zasílat do různých souborů a rozdělovat je například podle závažnosti a typu zprávy (například logy pro autorizaci `/var/log/auth.log` atd.).

Rotace logů (komprimování a zálohování starých záznamů) je prováděno programem *logrotate*, který zajišťuje rotaci logů po zvolených časových intervalech.

Nevýhody:

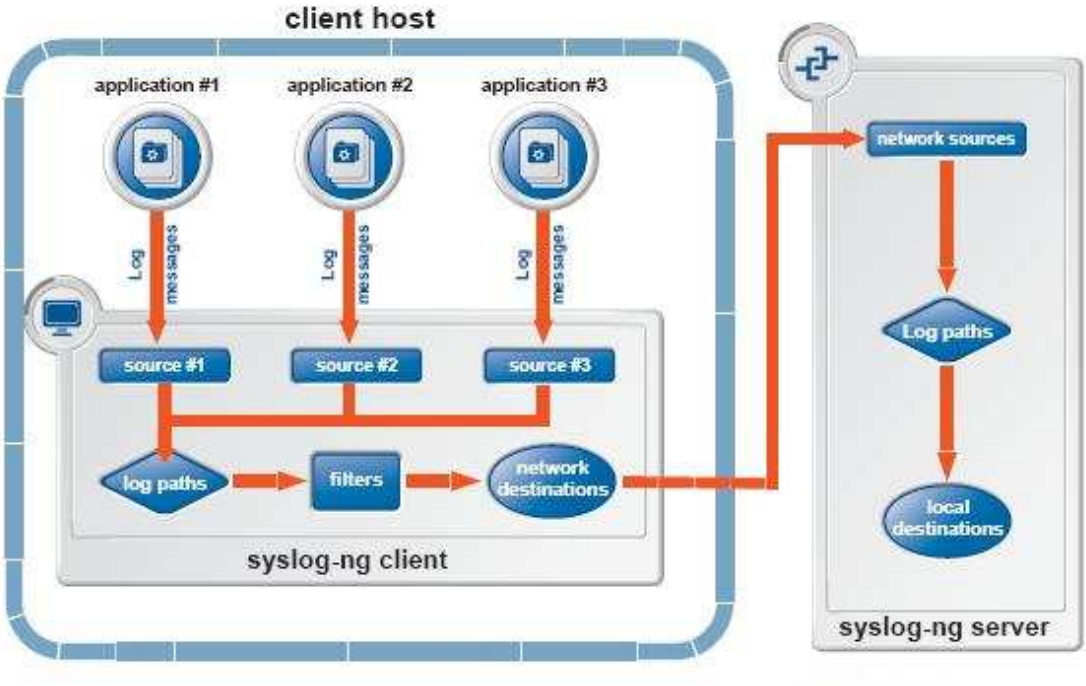
Slabinou syslogu je především logování přes síť pomocí protokolu UDP, který neposkytuje dostatečně bezpečnou a spolehlivou komunikaci. Možnou alternativou je syslog-ng, kterým umožňuje komunikaci jak přes UDP, tak přec TCP. V krátkosti se o syslogu-ng zmíníme v následující kapitole, přičemž nás bude zajímat konfigurace zdrojů, cílů a filtrů, možnost řízení toku zpráv a také možnost ukládání zpráv do databáze v syslog-ng Premium Edition.

5.1.2 Syslog-ng

Obecná charakteristika a způsob fungování:

Syslog-ng je logovací systém, který poskytuje tzv. centralizované logování. To znamená, že zprávy z jednotlivých zařízení, která jsou nazývána syslog-ng klienti, a na kterých je spuštěna aplikace syslog-ng, jsou přiváděny na centrální logovací server. Syslog-ng server zprávy následně třídí a ukládá. Viz následující obrázek.

[23] Pojmenovaná roura je struktura typu FIFO (*first in First out*) a zajišťuje komunikaci mezi dvěma procesy.



Obr. č. 8: Cesta logovací zprávy ze zdroje na klientovi do cílové destinace na serveru
(Zdroj: [28])

Na obrázku vidíme, že aplikace (*application* #1 až #3) zasílají do definovaného zdroje (*source* #1 až #3) v syslog-ng klientovi logovací zprávu. Zpráva může (ale nemusí) projít fází dalšího zpracování (filtrace[24], rozdělení[25], přepisování[26]). Pokud zpráva projde filtrací, je prostřednictvím síťového protokolu TCP zaslána do cílové destinace – na syslog-ng server. Syslog-klient poté začne zpracovávat další zprávu ze zdroje.

Syslog-ng server načte ze zdroje přijatou zprávu od syslog-ng klienta. Na zprávu následně, stejně jako je tomu na straně syslog-ng klienta, mohou být aplikovány výše uvedené operace a filtrační pravidla. Po té je zpráva uložena do konkrétní destinace a syslog-ng server začne zpracovávat další zprávu ze zdroje

Nyní uveďme obecně některé důležité vlastnosti aplikace syslog-ng. Některé z nich jsou dostupné pouze v syslog-ng Premium Edition, což bude vždy uvedeno v závorce jako (dostupné pouze v PE):

- Syslog-ng dokáže **sbírat zprávy** od jednotlivých klientů a zasílat je na syslog-ng server v **„heterogenním prostředí“**. To znamená, že dokáže sbírat zprávy z různých zařízení, kde jsou využívány **různé operační systémy a hardwarové platformy**. Syslog-ng může být přímo nainstalován v prostředí Linux, Unix, BSD, Sun Solaris, HP-UX, AIX a jako agent umožňující sběr zpráv také pod IBM i a Microsoft Windows.
- Využívá **k přenosu** zpráv mezi syslog-ng klienty a servery **TCP protokol**.
- Dokáže pracovat v síťovém prostředí **IPv4** i **IPv6**.

[24] Filtry jsou objekty, které provádějí selekci zpráv na základě zvolených kritérií (název aplikace, závažnost a podobně.)
[25] Rozdělování zpráv provádí *parsery*, které dokáží obsah jednotlivých zpráv segmentovat na pojmenované úseky.
[26] Přepisování provádí přepisovací pravidla, která jsou stejně jako parsery a filtry globálními objekty a mohou buď kompletně nahradit určitou část zprávy, nebo vyhledat nějaký řetězec nebo regulární výraz a nahradit pouze ten.

- Protože některé zprávy mohou obsahovat citlivá data, syslog-ng umožňuje **zakódování zpráv** pomocí **TLS** (*Transport layer security*) protokolu. (dostupné pouze v PE)
- **Pokud je spojení** se syslog-ng **serverem nedostupné**, tak aby nedocházelo ke ztrátám zpráv, jsou zprávy **ukládány na lokální disk** a odeslány teprve až je spojení se servrem obnoveno. (dostupné pouze v PE)
- Syslog-ng umožňuje **ukládat zprávy** jak **do** logovacích **souborů**, tak **i do databáze**, přičemž syslog-ng PE podporuje následující databáze: MSSQL, MySQL, Oracle, PostgreSQL, and SQLite.
- Zprávy je možné **ukládat v** uživatelem **zvoleném formátu**. (dostupné pouze v PE)
- V syslogu-ng je možné použít **filtraci zpráv**. Jedná se o implementovaná pravidla, která vybírají pouze některé zprávy. Například zprávy s určitou úrovní důležitosti, nebo zprávy pouze z některých aplikací.
- Syslog-ng umí **rozdělovat** zprávy **do pojmenovaných částí**, přičemž může tyto části zároveň **přepisovat**, a to **buď celé**, nebo **pouze** některé **řetězce** nebo **regulární výrazy**.

Syslog-ng používají především instituce a společnosti, které potřebují sbírat a řídit velké množství logů z mnoha zdrojů a centrálně je ukládat a organizovat. (např.: finanční instituce, datacentra, administrátoři serverových farem, společnosti poskytující webový, serverový nebo aplikační hosting a podobně.)

Konfigurace

Syslog-ng tedy pracuje tak, že se všechny zprávy ze všech zdrojů spojí do jedné fronty a následně postupují k části, kde jsou dále zpracovávány. Když zprávy projdou fází zpracování a filtrace, jsou odeslány do svých cílových destinací. Obsah logu tedy závisí na konfiguraci. Zdroje, cíle a části, kde dochází ke zpracování a filtrování zpráv, je možné prostřednictvím změny konfiguračního souboru syslogu-ng (`Syslog-ng.conf`) nastavit a uživatel má tak sám možnost určit, které zprávy odkud a kam ukládat.

Zdroje – syntax:

```
source identifier {source-driver(params); source-driver(params); ...
};
```

Zdroje mohou být: internal (interní zprávy syslogu-ng), unix-stream, unix-dgram (zprávy ze socketu stream nebo dgram), udp a tcp (zprávy z udp a tcp portu).

Příklad:

```
source s_all {
        internal();
        unix-stream("/dev/log");
};
```

Cíle – syntax:

```
destination identifier { destination-driver(params); destination-
driver(params); ... };
```

Cíle mohou být: udp, tcp, file (soubor), program

Příklad:

```
destination df_cron { file("/var/log/$YEAR/$MONTH/cron.log"); };
destination df_mail { file("/var/log/$YEAR/$MONTH/mail.log"); };
```

Filtry – syntax:

```
filter identifier { expression; };
```

Cíle mohou být: facility (kategorie), level (závažnost), program (jméno programu), host (jméno klienta), match (filtrování podle řetězce ve zprávě), filter (volání dalšího filtru).

Příklad:

```
filter f_cron { facility(cron); };
filter f_mail { facility(mail); };
```

Zdroje, cíle a filtry jsou následně spojeny:

```
log {
    source(s_all);
    filter(f_cron);
    destination(df_cron);
};
log {
    source(s_all);
    filter(f_mail);
    destination(df_mail);
};
```

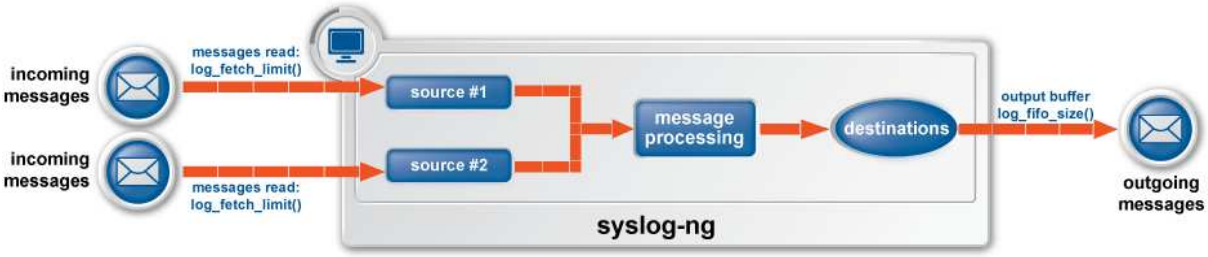
Makra

Pro lepší organizaci logů syslog-ng umí dynamicky vytvářet soubory, adresáře nebo názvy tabulek. K tomu využívá tzv. makra. Makra jsou identifikátory odkazující na nějakou část zprávy. Například makro *$HOST* odkazuje na jméno nebo IP adresu klienta, který zprávu poslal, nebo makro *$DAY* odkazuje na den v měsíci, kdy syslog-ng přijal zprávu. Použití maker v cestě cílové destinace logovacích souborů umožňuje například ukládání logů do souborů podle jména klienta, podle dnů, podle závažnosti atd. (např. `/var/log/FACILITY/$FACILITY.log`)

Řízení toku vstupních a výstupních zpráv (flow-control)

Aplikace syslog-ng periodicky kontroluje zdroje nadefinované v konfiguračním souboru. Když se objeví zpráva syslog-ng ji načte, případně upraví a uloží do výstupního bufferu (typu fifo), odkud jsou zprávy jednotlivě odesílány. Každá nadefinovaná destinace má svůj výstupní buffer, který má však jenom omezenou velikost a může dojít k jeho naplnění. Pokud je výstupní buffer plný, znamená to, že cílová destinace nemůže přijímat nové zprávy, proto v syslog-ng lze na základě nastavení parametru `log_fetch_limit()`, který může být nastaven globálně pro všechny zdroje nebo u každého zdroje zvlášť, fixně nastavit maximální počet (který by měl být menší než velikost výstupního bufferu) načtených zpráv během jednoho cyklu načítání zpráv ze zdrojů. Tento postup se nazývá řízení-toku zpráv (*flow-control*) a pokud je správně nastaven počet načtených zpráv s ohledem

na velikost výstupního bufferu. Tím je zamezeno ztrátě zpráv, ke kterým by k přetečení došlo[27]. Řízení toku zpráv ilustruje následující obrázek.



Obr. č. 9: Řízení toku zpráv v syslog-ng
(Zdroj: [28])

Možnost ukládání zpráv do databáze:

Zde v krátkosti načrtneme jakým způsobem syslog-ng PE (ve verzi 3.0) umožňuje ukládat záznamy do databáze.

V definování cílové destinace může být v syslog-ng PE také parametr `sql()`, který umožňuje definovat cílovou destinaci jako databázi.

```
sql(database_type host_parameters database_parameters [options]);
```

Jednotlivé parametry zahrnují parametry host (jméno nebo IP adresa) a databáze (typ, název, název použité databázové tabulky, názvy sloupců – v základním nastavení jsou to tyto sloupce: „datum“, „kategorie“, „závažnost“, „identifikace hosta“, „program“, „identifikace procesu“, „text zprávy“, a hodnoty, které se do jednotlivých sloupců mají zapsat a indexy v databázi pro rychlejší vyhledávání. Lze také nastavit maximální počet zpráv ve výstupní bufferu `log_fifo_size()`, časové pásmo atd.). Příklad zaslání zprávy do MS SQL databáze.

```
destination d_mssql {
sql(type(mssql) host("logserver") port("1433")
username("syslogng") password("syslogng") database("syslogng")
table("msgs_${R_YEAR}${R_MONTH}${R_DAY}")
columns("datetime varchar(16)", "host varchar(32)", "program varchar(32)",
"pid varchar(8)", "message varchar(4096)")
values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
indexes("datetime", "host", "program", "pid", "message"));
};
```

[27] V syslogu-ng 3.0 Premium Editon jsou zprávy, které by jinak byly ztraceny při přetečení výstupního bufferu, pokud cílová destinace není schopna zprávy přijímat, ukládány na lokální disk a poslána při obnovení přijímání zpráv cílovými destinacemi.

6 Logování ve spojení s Javou

V této kapitole se budeme zabývat možnostmi logování z aplikací nezávislé na operačním systému ve spojení s Javou,. Konkrétně budeme analyzovat utilitu Log4j. Jako zdroj informací byly použity materiály [6], [9], [16], [21].

6.1 Nástroje pro logování v aplikacích

Pro odladění chyb v aplikacích a sledování jejich činnosti lze obecně použít dvě kategorie nástrojů. Tzv. debuggery a loggery. Pomocí debuggerů lze najít chyby v programovém kódu, nicméně je nutné aplikaci znovu překompilovat. Druhým nástrojem jsou loggery, které mají oproti debuggerům výhodu, že mohou pracovat za běhu aplikací, bez nutnosti aplikace znovu kompilovat. Přičemž parametry logování lze nastavovat bez vlivu na aplikace v externích konfiguračních souborech loggerů. Logováním v aplikacích budeme rozumět zaznamenávání posloupnosti událostí, které v daných aplikacích probíhají. Logování z aplikací se vytváří pomocí vládání logovacích příkazů do zdrojového kódu aplikace. Tyto příkazy pak vedou k vytvoření logovacích zpráv, které jsou následně podle konfigurace formátu zprávy a cílové destinace ukládány a vytváří tzv. aplikační log.

Nástroje pro logování z aplikací, konkrétně těch, které podporují Javu, existuje několik. Uveďme některé z nich: Java logging API, Log4j, Logging Toolkit for Java**,** Java Logging Framework**,** SLF4j, Craftsman spy, Lumberjack, LimpidLog. V následující kapitole se budeme zabývat jedním z nich: Log4j.

6.2 Log4j

6.2.1 Obecná charakteristika

Jedním z frameworků pro logování z aplikací je API balíček Log4j. Log4j je *open source* projekt Apache software foundation[28] pod licencí Apache, který se stal standardem pro logování v Javě a byl přenesen i do dalších programovacích jazyků (C, C++, Python, PHP).

Samotné logování s definováním formátu zprávy a jejím cílovým umístěním zajišťují v Log4j především tyto třídy:

`Logger`: Třída logger, která je zodpovědná logování a jeho řízení jako celek.
`Appender`: K třídě `Logger` se připojuje třída `appender`, která určuje cílovou destinaci, kam se budou logované zprávy ukládat.
`Layout`: Třída `Layout` zajišťuje formátování zprávy.

Blíže se budeme jednotlivým třídám věnovat ještě dále.

[28] Decentralizovaná komunita vývojářů vytvářející *free* a *opensource* software pod licencí Apache.

Úrovně závažnosti

V Log4j je rozsah úrovní logovaných zpráv definován ve třídě `Level`. Přičemž Log4j pracuje s těmito úrovněmi závažnosti zprávy:

- `OFF`: logování je vypnuto
- `DEBUG`: zprávy ladění
- `INFO`: informační zprávy
- `WARN`: zprávy varování
- `ERROR`: zprávy chyb
- `FATAL`: zprávy kritických chyb
- `ALL`: logují se všechny zprávy

6.2.2 Logger

Každá třída aplikace může mít svůj vlastní logger. Pokud třída nemá vlastní logger použije se logger nejbližšího předchůdce. Pokud žádný předchůdce nemá vlastní logger použije se kořenový logger. Vytvoření instance kořenového loggeru:

```
Logger logger = Logger.getRootLogger();
```

a nastavení minimální úrovně pro logování pomocí metody setLevel:

```
logger.setLevel((Level)Level.WARN);
```

6.2.3 Appender

Appender je třída, která se stará o směrování výstupu logovaných zpráv. Appenderů je několik druhů například:

- `FileAppender`: `Ukládání výstupu do souboru.`
- `ConsoleAppender`: Výpis zpráv přímo na konzoli nebo obrazovku
- `SocketAppender`: Posílá logy na vzdálený logovací server.
- `JDBCAppender`: Zápis logů do databáze.
- SyslogAppender: Posílá zprávy syslog démonu.

Příklad použití `FileAppender`:

```
FileAppender appender = null;
try {
      appender = new FileAppender(new PatternLayout(),
"jméno_souboru");
      } catch(Exception e) {}
```

`JDBCAppender` – Umožňuje ukládání zpráv do databáze. Příkazy pro zápis do databáze se nastavují pomocí `setSQL`, jejímž parametrem je řetězec, např.

```
INSERT INTO JDBCTEST (Date, Logger, Priority, Message) VALUES ('%d', '%c',
'%p', '%m')
```

6.2.4 Layout

Třída Layout je určená pro formátování logovacích zpráv. Log4j má 4 základní druhy Layoutů:

`PatternLayout`: Formátování podle transformační masky.
`HTMLLayout`: Logovací zprávy jsou uloženy v tabulce v HTML kódu.
`SimpleLayout`: Nastavení formátu zprávy: chyba – logovaná zpráva.
`XMLLayout`: Formátování logovaných zpráv do XML souboru.

Příklad použití appenderu (`FileAppender`) a layoutu (`SimpleLayout`):

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;
import org.apache.log4j.FileAppender;

public class LogovanaTrida {
	static Logger logger = Logger.getLogger(LogovanaTrida.class);
	public static void main(String args[]) {
		SimpleLayout layout = new SimpleLayout();

		FileAppender appender = null;
			try {
				appender = new
				  FileAppender(layout,"vystup.txt",false);
				} catch(Exception e) {}

				logger.addAppender(appender);
				logger.setLevel((Level) Level.DEBUG);

				logger.debug("Úroveň DEBUG");
				logger.info("Úroveň INFO");
				logger.warn("Úroveň WARN");
				logger.error("Úroveň ERROR");
				logger.fatal("Úroveň FATAL");
			   }
			}
```

6.2.5 Konfigurace

Konfigurace Log4j s parametry logování je nastavena v souboru `log4j.properties`, ale může být definována v XML nebo textovém souboru. Konfigurace textovém souboru se připojí pomocí metody `PropertyConfigurator`,
`PropertyConfigurator.configure("konfigurace.txt");`

Konfigurace v XML souboru se připojí pomocí:
`DOMConfigurator.configure("configurationfile.xml");`

7 Definice vlastností obecného logovacího systému za použití principů z AS/400

V této kapitole obecně popíšeme některé důležité vlastnosti, které by navrhovaný logovací systém měl splňovat, přičemž inspirací při návrhu logovacího systému nám bude v některých aspektech systém zpráv na AS/400 – především pak úroveň předdefinovaných zpráv, kde jsou jednotlivé zprávy uložené ve jmenném prostoru (souborech zpráv), obsahují vlastní identifikaci a již vytvořený parametrizovaný text umožňují práci s proměnnými. Bližší popis systému zpráv na AS/400 obsahuje kapitola *4.4 Systém zpráv na AS/400*.

Účel logování:

Při návrhu vlastností logovacího systému musíme především vycházet z toho, k čemu jsou obecně záznamy logu používány. Je to především sledování činností zařízení, aplikací a uživatelů v čase, přičemž analýza logu může také napomoci ke sběru informací, které jsou využívány k vylepšení aplikací nebo pro zvýšení bezpečnosti systému. Více o účelu logování obsahuje kapitola *3 Logování zpráv*.

Log principiálně obsahuje zprávy z různých zdrojů informující o různých událostech poskládaných v časovém sledu za sebou. Množství zpráv, které se ukládají do logu však může dosahovat velmi vysokého počtu, čímž je, pokud chceme sledovat pouze určité události, snížena přehlednost a orientace v logovacích záznamech. Jestliže chceme zvýšit „čitelnost“ a možnost následné analýzy logu, je zapotřebí navrhnout obecný logovací systém tak, aby bylo možno na základě konkrétních kritérií **záznamy logu selektovat** a vybírat pouze zprávy o událostech, které pro nás aktuálně mají nějaký význam.

Identifikace zpráv:

Jak již bylo řečeno, log může obsahovat velké množství zpráv, přičemž jednotlivé zprávy se mohou v logu opakovat. Pokud chceme v logu vyhledávat či nad ním potencionálně provádět dotazování (pokud bychom log ukládali do databáze), musí být jednotlivé zprávy v logu nějakým způsobem identifikovatelné.

Tuto identifikaci lze provést přiřazením jedinečného identifikátoru (čísla nebo kódu z písmen a čísel) každé zprávě. V AS/400 je tato identifikace prováděna přiřazením kódu, který je unikátní v daném jmenném prostoru (souboru zpráv). Identifikátor se skládá ze 3 písmen a 4 čísel např. `USR4310`

Samotná identifikace zprávy prostřednictvím přiřazení jedinečného kódu (ID zprávy) však poskytuje možnost vybírat z logu pouze zprávy s daným kódem.

Závažnost zpráv:

Jedním z prostředků selekce (respektive filtrování) zpráv, používaný i v ostatních systémech (viz AS/400, syslog, Log4j), je definice závažnosti či priority zpráv rozdělující zprávy na několik úrovní, které mají rozdílnou váhu. Takové úrovně mohou rozlišit zprávy, které pouze o něčem

informují nebo varují před možnou komplikací, od těch, které hlásí závažnou chybu a podobně. Definice závažnosti je tedy jedním z možných kritérií výběru zpráv z logu.

Definování závažnosti nám však může poskytnout pouze výběr zpráv určité úrovně závažnosti (například si můžeme nechat vypisovat pouze zprávy informující o závažných chybách), ale nemůže poskytnout sledování určité konkrétní aplikace nebo uživatele, o němž v logu existují konkrétní záznamy, které jsou však „ukryty v nánosu“ záznamů dalších, informujících o jiných aplikacích a uživatelích.

Parametrizace textu zprávy:

Parametrizace textu zprávy je dalším prostředkem jak podle hodnot parametrů v logu vyhledávat konkrétní informace. Za proměnnou zprávy bude možné dosadit hodnotu definovaného typu (viz proměnné zprávy v AS/400 kapitola *4.4.4 Předdefinované zprávy*), která vyjadřuje identifikaci uživatele, souboru, či jakoukoliv jinou informaci. Například :

```
12-12-08 11:05 Uživatel 1234 byl odhlášen.
12-12-08 11:10 Program xyz byl spuštěn.
```

Parametry budou naplněny při vytváření zprávy (analogicky jako je tomu v příkazu pro zasílání zpráv na AS/400, kde jsou hodnoty parametrů vkládány v parametru `MSGDTA()`). Hodnoty parametrů spolu s identifikací zprávy budou následně uloženy do logu.

Na základě hodnot parametrů by pak bylo možné vyhledávat (provádět dotazování prostřednictvím příkazů `select`) v logu zadáním podmínky v klauzuli `where`, kde se hodnoty parametrů rovnají určité zadané hodnotě. Systém by v logu našel výskyty, kde jsou tyto hodnoty obsaženy, přičemž by vyzvedl z úložného prostoru zprávy podle odpovídajícího kódu a následně je zobrazil. Výsledek by pak obsahoval selekci logu, kde by byly vypsány pouze zprávy, v kterých jsou obsaženy zadané hodnoty parametrů.

Např. vybrat všechny zprávy, které se týkají uživatele s uživatelským jménem „Miroslav“.

```
10-12-08 15:05 Uživatel Miroslav byl přihlášen.
10-12-08 17:45 Uživatel Miroslav byl odhlášen.
```

Parametrů v rámci zprávy by může být také více. Například:

```
10-12-08 10:45 Uživatel Miroslav smazal soubor Test.
```

Je však důležité, aby návrh struktury zprávy s použitím proměnných v textu zprávy počítal, a aby byla vytvořena standardizace těchto zpráv. Tedy to, že zpráva má určitý formalizovaný text, který se vztahuje k danému parametru (respektive k tomu co má být jeho obsahem – jméno souboru, číslo klienta a podobně.), do kterého může být dosazena hodnota proměnné určitého typu. Tento typ by mohl být uživatelsky definován a někde externě uložen v seznamu uživatelsky definovaných datových typů.

Předdefinování zpráv:

Standardizaci zpráv lze vytvořit jejich předdefinováním, přičemž někde v systému (např. v XML souboru nebo v databázové tabulce) bude zpráva s textem již uložena a jednotlivé zprávy budou od sebe jedinečně odlišeny vlastním jedinečnou identifikací (kódem) v rámci svého jmenného prostoru (souboru zpráv nebo databázové tabulce).

Do logu by nebyly ukládány zprávu s textem, ale pouze jejich identifikace a hodnoty parametrů, které jsou součástí textu zprávy. Text zprávy v logu bude „vyzvednut“ a použit až při zobrazování logu a za proměnné budou dosazeny konkrétní hodnoty.

Přístup k logu:

Přístup k logu by měl být z bezpečnostních důvodů autorizován, přičemž by měla být vytvořena autorizace pro více typů uživatelů podle určitého stupně oprávnění. Uživatel s nejnižším stupněm oprávnění by měl možnost log pouze číst nebo v něm případně i vyhledávat. Uživatel s nejvyšším stupněm oprávnění by pak měl také možnost manipulace se zprávami a úpravy logu.

8 Analýza požadavků a návrh systému

8.1 Funkční požadavky

8.1.1 Obecná definice systému

Jedná se o obecný systém logování zpráv pro doménové objekty, který zajišťuje přidávání logů (zpráv) k doménovým objektům v rámci logované aplikace, ke které je tento systém připojen jako logovací subsystém (komponenta), přičemž jednotlivé záznamy logu jsou ukládány do databáze.

Obecnost logovacího systému spočívá v jeho nezávislostí na konkrétním operačním systému a nezávislostí na konkrétní aplikaci, ve které má být logování prováděno.

Tento systém je realizován pro firmu Aegis s.r.o. a podle požadavků firmy má být systém vytvořen na základě principů používaných v systému zpráv na AS/400. Konkrétně:

- Samotný log obsahuje pouze identifikaci předdefinované zprávy a hodnoty parametrů.
- Zprávy jsou předdefinovány a uloženy v souboru nebo databázi.
- Každá zpráva má jednoznačnou identifikaci.
- Součástí předdefinované zprávy je text umožňující práci s proměnnými, které jsou naplněny konkrétními hodnotami při vytváření zprávy.
- Zprávy mají různé úrovně závažnosti.

Systém by měl být následně integrován do aplikačního prostředí firmy Aegis s.r.o. , přičemž zdroje zpráv budou reálné doménové objekty, které firma ve svých aplikacích používá.

8.1.2 Požadavky na systém

Ze strany firmy Aegis s.r.o. jsou konkrétní požadavky na navrhovaný systém následující:

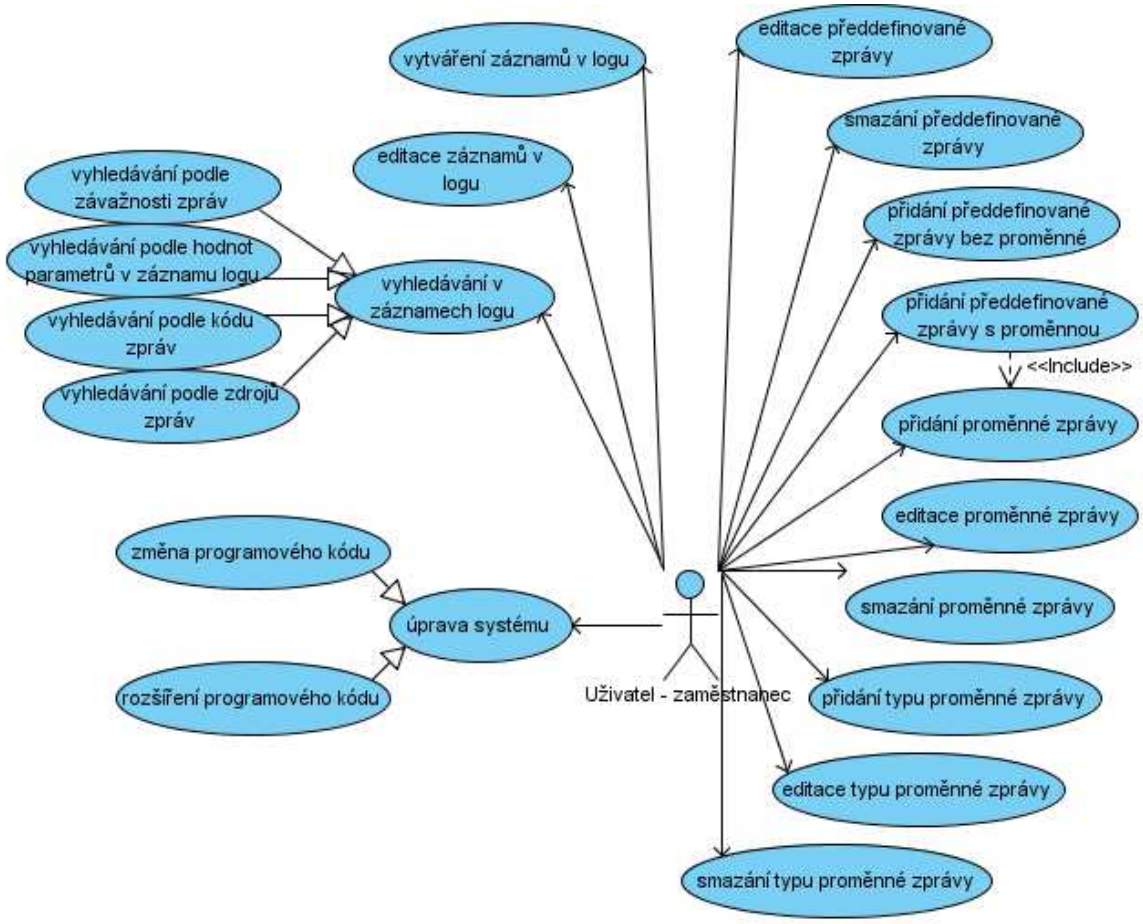
Implementace systému logování zpráv v jazyce Java, který bude umožňovat:

- Definici zpráv
  - ID zprávy
  - Text zprávy a jeho formátování
  - Proměnné zprávy
  - Další vlastnosti jako závažnost atd.
- Zasílání zpráv
  - Programové vybavení pro zapsání zprávy do logu/fronty zpráv
  - Nastavení parametrů zaslané zprávy
- Práce se zprávami
  - Vyhledání zpráv v logu

8.1.3 Typy uživatelů systému

Uživatelé systému jsou zaměstnanci firmy Aegis, kteří mohou přistupovat k záznamům logu a dále s ním pracovat: vyhledávat v logu, případně záznamy v logu upravovat. Současně také mohou do systému zadávat další předdefinované zprávy a proměnné editovat je. Sami také mohou systém

dále upravovat a rozvíjet. Existovat tak bude pouze jeden typ uživatele (uživatel-zaměstnanec), který může zároveň vystupovat také v roli administrátora systému. Následující obrázek reprezentuje use-case diagram, kde jsou jednotlivé případy použití uvedeny.



Obr. č. 10: Funkčnost aplikace z pohledu uživatele – zaměstnance

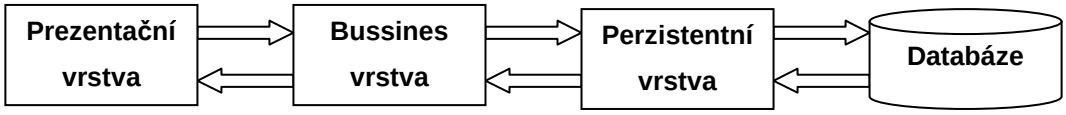
8.1.4 Nefunkční požadavky

- implementace v jazyce Java
- data budou ukládána do databázového prostoru MS SQL, který firma používá
- pro ukládání dat do databáze bude použito rozhraní JDBC

8.1.5 Architektura systému

Návrh architektury bývá zpravidla tvořen z několika vrstev, které mají hierarchickou strukturu. Rozlišovány jsou především 3 základní vrstvy:

- Perzistentní vrstva (zajišťuje uložení dat v perzistentní podobě v souborech nebo databázi)
- Bussines vrstva/aplikační vrstva (zajišťuje zpracování dat a logiku aplikace)
- Prezentační vrstva (zajišťuje zobrazení dat a rozhraní pro komunikaci uživatele a aplikace)



Obr. č. 11: Základní vrstvy používané pro návrh systému

V našem případě se bude jednat o poměrně malé softwarové dílo a systém bude tvořen pouze ze dvou vrstev, a to z vrstvy perzistentní a aplikační. **Prezentační** vrstva není zadavatelem vyžadována a **není v rámci této práce implementována**. Zobrazování logu a práce se záznamy, předdefinovanými zprávami a proměnnými bude uživatel provádět přímo pomocí SQL příkazů nad databází.

Popis systému:

Logovací systém je vytvořen jako balíček (`cz.aegis.util.log`), který lze připojit jako komponentu k Java aplikaci, ve které má probíhat logování. Záznamy logu jsou ukládány do databáze, kde jsou také uložené příslušné předdefinované zprávy a proměnné.

Jeho aplikační logiku zajišťují rozhraní `businesLogger` a `businessLoggerManager` implementované třídami `businesLoggerImpl` a `businessLoggerManagerImpl`, které pracují s DAO (*Data Acces Object*) – ty zajišťují přístup k databázi (perzistenční vrstva).

DAO (jednotlivé DAO můžeme vidět na obrázku 13 a 14) poskytují rozhraní, která jsou vždy implementovány příslušnými třídami využívající JDBC. Například rozhraní `MessageDao` implementuje `MessageDaoJDBC`, rohraní `LogEntryDao` implementuje `LogEntryDaoJdbc` atd.

Přes rozhraní `businesLogger` je možné vytvářet záznamy v logu k daným doménovým objektům aplikace, na které se logovací systém váže pomocí `tgt_id`. Podmínkou je, že každý doménový objekt má unikátní `id` v rámci celé aplikace.

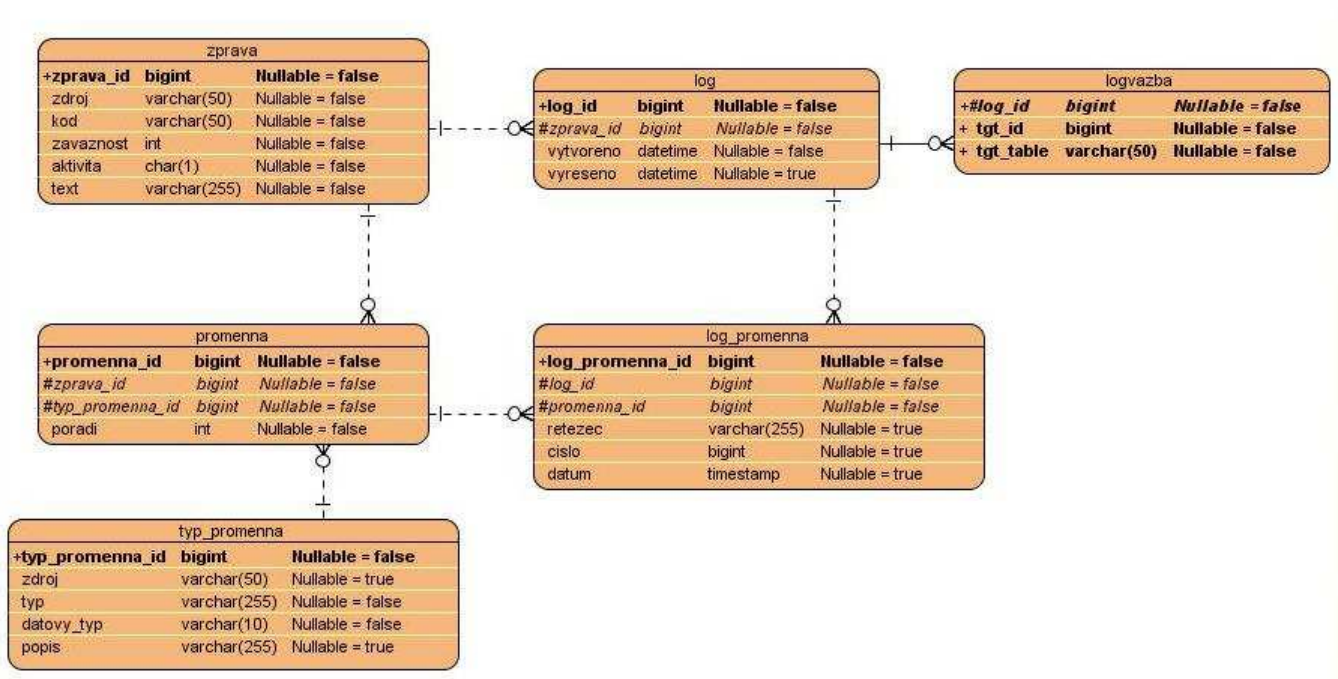
Logovací systém také obsahuje programové vybavení (třídy a rozhraní) pro vytváření předdefinovaných zpráv, proměnných zprávy a typů zprávy a jejich uložení v databázi, to zajišťuje `businessLoggerManagerImpl`.

Součástí logovacího systému jsou také testovací třídy (adresář `src/test/java`), které při spuštění testu provedou pomocí implementovaných tříd zápis testovacích dat do databáze a vytvoří testovací záznamy ve všech tabulkách databáze.

8.1.6 Databáze a perzistentní vrstva

Perzistentní vrstva bude využívat aplikačního rámce Spring a návrhového vzoru objektů přistupujícím k datům – DAO, které budou mít přes rozhraní JDBC na starost komunikaci s databází, čímž budou odstiňovat aplikační vrstvu od přímé manipulace s úložištěm dat. Přístup k databázi bude definován v elementu (*beaně*) v konfiguračním XML souboru, takže pro změnu připojení k databázi bude třeba přepsat pouze příslušný element bez nutnosti dalších změn v aplikaci.

Strukturu databáze a vztahy mezi doménovými objekty reprezentuje následující ER diagram.



Obr. č. 12: ER diagram

Vazební tabulka:

Každý záznam v logu se váže ke konkrétnímu doménovému objektu v aplikaci, kde je logovací systém používán. Tabulka `logvazba` je vazební tabulkou, která zajišťuje vztah many-to-many mezi logovanou zprávou a objekty, ke kterým se záznam v logu vztahuje. Podmínkou je, že každý doménový objekt v rámci celé aplikace (nikoliv pouze v rámci tabulky) jednoznačné `id` na které odkazuje `tgt_id` v tabulce `logvazba`. `tgt_table` obsahuje název tabulky doménového objektu v databázi (např. Klient). Každý doménový objekt aplikace tak může mít n záznamů v logu a každý záznam se může vztahovat k n objektům.

Log:

Jednotlivé záznamy v logu obsahují vlastní identifikaci (`log_id`) časové razítko kdy byl záznam vytvořen (`vytvoreno`) a časové razítko kdy byla daná událost vyřešena (`vyreseno`). Pokud nebyla vyřešena obsahuje hodnotu `null`. Jako cizí klíč je v záznamu logu uloženo `id_zpravy`, které vytváří vazbu na konkrétní předdefinovanou zprávu.

Hodnoty parametrů v logu:

Hodnoty parametrů, které patří ke konkrétnímu záznamu v logu, jsou uloženy v tabulce `log_promenna`. Každá hodnota parametru (identifikovaná `log_promenna_id`) náleží ke konkrétnímu záznamu v log (`log_id`) a jedná se o konkrétní proměnnou zprávy (`promenna_id`). Hodnoty parametrů mohou být pak buď číslo (`cislo`), řetězec (`retezec`) nebo datum (`timestamp`).

Předdefinované zprávy:

Předdefinované zprávy jsou uloženy v tabulce `zprava`. Obsahují:

`id_zpravy` – identifikace zprávy v databázi
`zdroj` – události jsou rozděleny do zdrojů (například aplikace Centrálního registru úvěrů, Insolvenční registr)
`kod` – identifikace zprávy, každá zpráva má unikátní kód v rámci zdroje
`zavaznost` – závažnost zprávy
`aktivita` – aktivita určuje, zda je zpráva aktivní (a/n)
`text` – text zprávy

Proměnné zprávy:

Zprávy mohou obsahovat proměnné tabulka `promenna`, které každá mají svoji identifikaci (`id_promenne`), náleží konkrétní zprávě (`id_zpravy`) a jsou určitého typu (`typ_promenna_id`). Ve zprávě může být více proměnných, proto je zde určeno pořadí (`poradi`) v jakém se v textu zprávy nachází.

Typ proměnné:

Každá proměnná je určitého typu (`typ_promenna_id`), patří do určitého zdroje (`zdroj`) jsou určitého datového typu (`datovy_typ`) a jsou definovány jako uživatelský datový typ (`typ`) označený jedinečným kódem (např. `KLIENTID`) v rámci zdroje. Mohou také obsahovat také bližší popis (`popis`) typu proměnné.

8.1.7 Identifikace zprávy, formát textu zprávy, závažnost zprávy

Identifikace zprávy:

Jednoznačná identifikace zprávy je kombinace: **kód zprávy** a **zdroj**. Kód zprávy je jedinečný v rámci zdroje – konkrétní aplikace, ve které probíhá logování. Podobně jako je tomu u AS/400, kde kód zprávy je unikátní v konkrétním souboru zpráv.

Formát textu zprávy:

Text zprávy může obsahovat libovolné znaky. Zpráva může obsahovat také proměnné. V AS/400 jsou v textu pro proměnné použity znaky &1 až &n. V našem případě vložíme na pozici v textu, kde se bude vyskytovat proměnná, zástupný znak %%.

Při zobrazování zprávy by za zástupný znak byla dosazena hodnota proměnné zprávy. Pokud by proměnných bylo v textu více, budou do textu zprávy za zástupné znaky postupně dosazovány podle pořadí (`poradi` v tabulce `promenna`).

Závažnost zprávy:

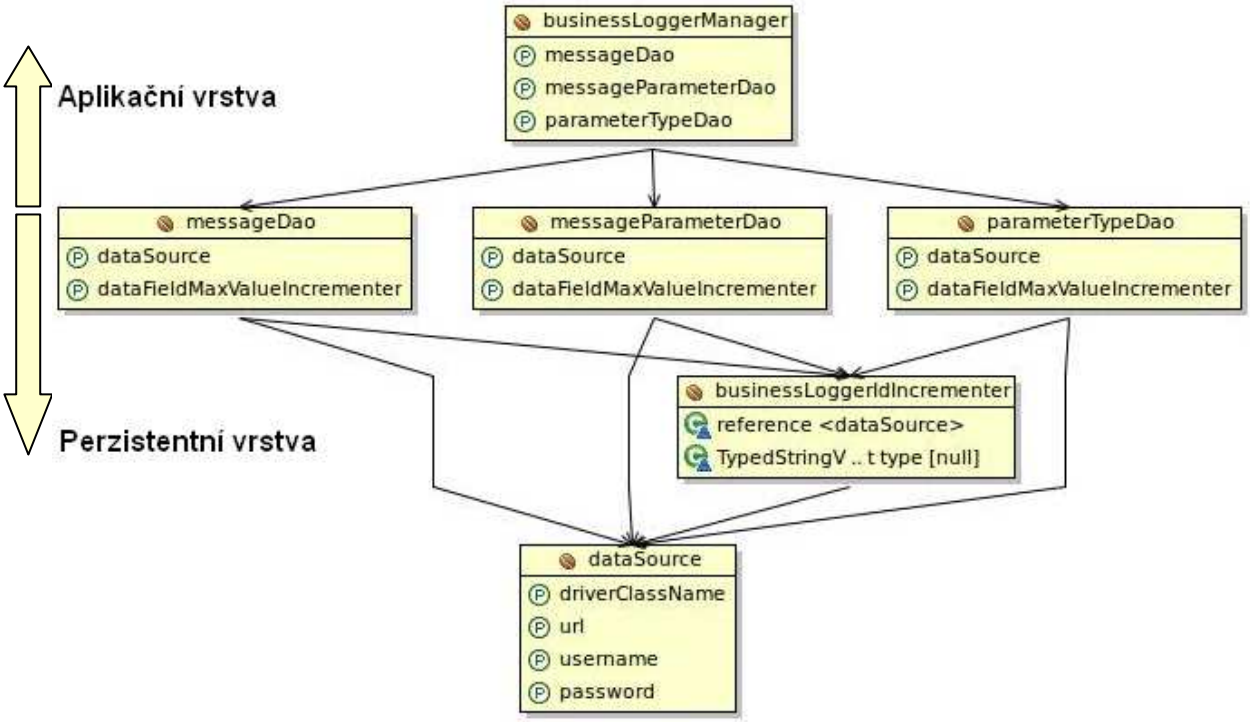
Závažnost zprávy je v databázi uložena jako libovolné číslo integer. I když lze zapsat úroveň závažnosti jako jakékoliv číslo, byla po dohodě se zadavatelem vytvořena konvence pro používání těchto úrovní závažnosti:

```
NULL(0), DEBUG(10), INFO(20), WARNING(40), ERROR(60)
```

8.1.8 Bussiness/aplikační vrstva

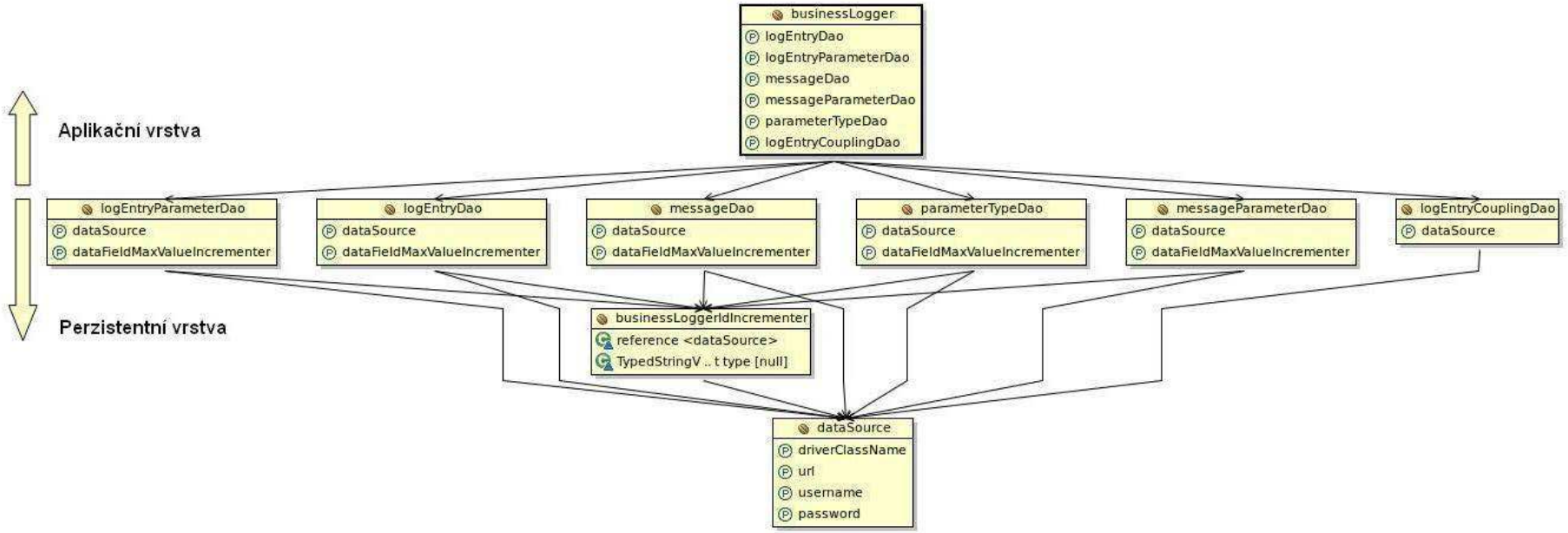
Aplikační vrstva zajišťuje logiku aplikace. V našem případě je pro logování záznamů k doménovým objektů vytvořeno rozhraní `businesLogger` a `businessLoggerManager`, implementované třídami `businesLoggerImpl` a `businessLoggerManagerImpl`, které zajišťují logování jako celek. V těchto třídách však není implementována přímá manipulace s databází, to zajišťují v perzistentní vrstvě namapované jednotlivé DAO objekty.

Závislosti jednotlivých objektů v aplikační a perzistentní vrstvě ilustrují následující obrázky.[29]



Obr. č. 13: Závislosti objektů a rozdělení vstvev – businessLoggerManager

[29] Obrázky jsou vytvořeny pomocí Spring IDE a popisují vazby mezi jednotlivými objekty (*beanami*), které jsou konfigurované Spring frameworkem.



Obr. č. 14: Závislosti objektů a rozdělení vrstev – businessLogger

9 Implementace a testování aplikace

Implementace logovacího systému byla vytvářena přímo v prostředí firmy Aegis s.r.o. pod vedením Ing. Tomáše Vítka. Jako podklady a zdroje pro implementaci byly použity [6], [18], [19], [20], [23], [25], [26], [29], [31].

9.1 Použité nástroje

9.1.1 UML

Jazyk UML (*Unified Model Languague*) je unifikovaný modelovací jazyk s grafickou sémantikou a syntaxí. Jedná se tedy o grafický jazyk pro specifikaci, vizualizaci, konstrukci a dokumentaci prvků vyvíjené aplikace. Jeho základními elementy jsou:

- Předměty – jedná se o uzavřené plošné tvary (kružnice, obdélníky kvádry a podobně) reprezentující takzvané strukturní abstrakce. Strukturní abstrakce jsou programové třídy, aplikační či objektové rozhraní, případy užití, komponenty či uzly. Dalšími strukturními abstrakcemi jsou tzv. chování (reprezentují komunikace mezi jednotlivými objekty modelu – různé šipky a propojovací čáry), seskupení (graficky seskupuje části diagramu na nižší úrovni) a poznámky, které blíže popisují vlastnosti a chování elementů modelu.

- Relace – reprezentují vztahy mezi jednotlivými elementy.Typy relací jsou Asociace (obecná souvislost předmětů), závislost (změna předmětu způsobí změnu jiného předmětu), generalizace (jeden předmět je specializací jiného předmětu) a realizace (dohoda, za jejíž splnění je zodpovědný jiný předmět).

- Diagramy – diagramy zachycují různé pohledy na aplikaci nebo její části (např. diagram případů použití, diagram tříd, diagram aktivit, stavový diagram atd.)

V našem případě byl pro modelování případů použití (obr. 10) použit nástroj Visual Paradigm for UML 6.4 Enterprise Edition.

Tento nástroj byl také použit k vytvoření ER diagramu (*Entity Relationship Diagram*)[30], který reprezentuje model databáze (obr. 12).

Z vytvořeného ER diagramu byl vygenerován skript jako *create script* v databázi MS SQL používanou ve firmě Aegis. Databáze je ve firmě používána ve verzi MS SQL Server 09.00.1399.

Soubor pro Visual Paradigm for UML 6.4 Enterprise Edition s modelem případů použití a ER diagramem (`log_system.vpp`) a vygenerovaný *create script* (soubor `create_db.sql`) je uložen na přiloženém CD v adresáři `database`.

[30] ER diagram není diagram UML.

9.1.2 Jazyk Java

Jazyk Java je objektově orientovaný programovací jazyk. Byl vytvořen firmou Sun Microsystems, která v roce 1995 uvedla jeho první verzi. Existují platformy Javy pro různá zabudovaná zařízení Java Micro Edition (Java ME), čipové karty Java Card, aplikace pro desktopové počítače Java Standard Edition (Java SE) i pro distribuované systémy pracující na počítačích propojených pomocí internetu Java Enterprise Edition (Java EE).

Java je pro v současné době jedním z nejpoužívanějších programovacích jazyků. Jejími charakteristikami jsou především:

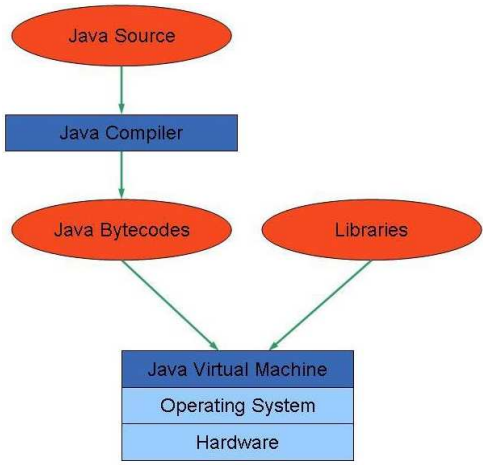
- Přenositelnost a nezávislost na architektuře – program vytvořený v Javě v podobě zdrojového i přeloženého kódu je přenositelný mezi různými operačními systémy (podmínkou je existence virtuálního stroje Javy – *Java Virtual Machine* pro danou architekturu).

- Objektová orientace – všechny datové typy jsou objektem (kromě několika primitivních datových typů).

- Jednoduchost – syntaxe je zjednodušenou verzí C a C++.

- Robustnost a bezpečnost – pomocí mechanismu výjimek, jejichž obsloužení si překladač vynucuje, ošetřuje chybové stavy v programu. Poskytuje automatickou typovou kontrolu a také správu paměti, z které jsou automaticky odstraňovány nepotřebné objekty.

- Interpretovatelnost – namísto do strojového kódu je kód překládán do mezikódu (bajt kód), který je nezávislý na architektuře. Program pak může pracovat na libovolném počítači nebo zařízení, který disponuje virtuálním strojem Javy, který ho interpretuje.



Obr. č. 15: Kompilace a provedení programu v Javě
Zdroj [26]

Pro vývoj aplikace v jazyce Java bylo použito vývojového prostředí Eclipse Platform.

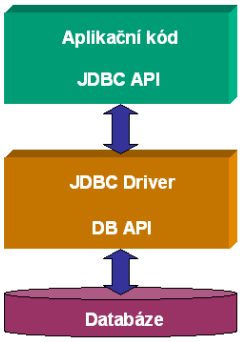
Eclipse Platform

Eclipse je *open source* vývojová platforma určená pro programování v jazyce Java. Pomocí pluginů[31] může podporovat i programování v jiných programovacích jazycích (C++ nebo PHP). V základní verzi obsahuje Eclipse editor zdrojových textů i vlastní kompilátor a debugger. Základní verzi lze stáhnou na oficiálních stránkách projektu http://www.eclipse.org. Při tvorbě aplikace bylo použita Eclipse Platform 3.4.0.

9.1.3 JDBC

JDBC (*Java Databáze Conectivity*) je technologie, která poskytuje aplikační rozhraní (API) pro přístup k databázi v aplikacích Javy. JDBC však není určeno pouze pro přístup k relačním databázím, ale může pracovat s jakýmkoliv daty, která jsou strukturovaná do tabulek.

Základem funkčnosti JDBC je ovladač, který je zpravidla vytvořen přímo výrobcem konkrétního databázového systému a je dostupný volně ke stažení z internetu. Ovladač zajišťuje přístup do databáze a její nativní volání, od kterého je programátor odstíněn. Výhodou koncepce JDBC je, že programátor může používat jednotné rozhraní JDBC API pro libovolnou databázi. Spojení s ní pak zajišťuje konkrétní ovladač. Architekturu JDBC ilustruje následující obrázek.



Obr. č. 16: Architektura JDBC
Zdroj [25]

Přístup do databáze pomocí JDBC:

Použití JDBC se sestává z několika kroků:

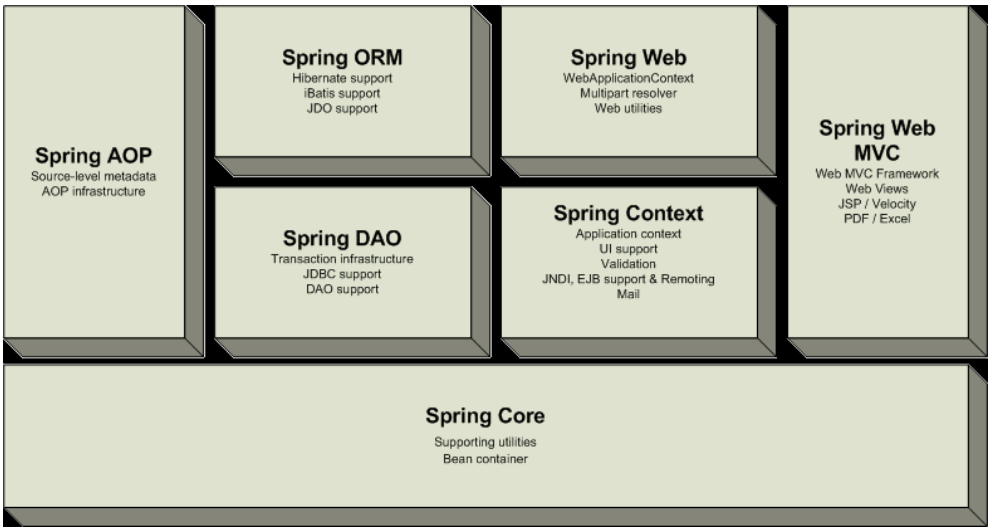
1) Načtení JDBC ovladače – načítáme třídu pro JDBC ovladače.
2) Alokace objektu `Connection` – tento objekt reprezentuje spojení s databází.
3) Alokace objektu `Statement` – objekt reprezentuje příkaz pro databázi (standardně SQL dotaz)
4) Provedení dotazu pomocí `Statement` objektu – Operace vrátí množinu výsledků v objektu typu `ResultSet`.
5) Zpracování výsledku – Projdou se všechny řádky z `ResultSet` a požadovaným způsobem se zpracují.
6) Uzavření objektů `ResultSet`, `Statement`, `Connection`.

[31] „Zásuvný modul“ je software, který pracuje jako doplňkový modul jiné aplikace a rozšiřuje tak její funkčnost.

9.1.4 Spring Framework

Spring je *open source* Java aplikační framework distribuovaný pod licencí Apache 2.0. Poslední verze (v době psaní této práce – leden 2009) je *Spring Framework 2.5.6*. Motivem pro jeho vytvoření je usnadnění vývoje Java aplikací.

Spring poskytuje poměrně širokou škálu technologií. Důležitým konceptem je návrhový vzor *Inversion of Control/Dependency Injection*, který podporuje nezávislost jednotlivých vrstev aplikace, přičemž přesouvá odpovědnost za vytvoření vazeb spolupracujících objektů z aplikace na framework. Jeho vlastností také je, že je modulární, to znamená, že můžeme používat pouze jen určitou jeho část, která se nám při vývoji aplikace bude hodit. Jednotlivé moduly Springu reprezentuje následující obrázek.



Obr. č. 16: Základní moduly Spring
Zdroj [29]

V našem případě využijeme především modul DAO, který poskytuje abstraktní vrstvu pro práci s JDBC API umožňující některá zjednodušení při tvorbě kódu (odstraňuje opakování se kódu při získání `Connection`, vytváření `Statementu` a procházení `ResultSet`), a také Spring Context, který pomocí aplikačního kontextu vymisťuje vazby mezi vrstvami aplikace do konfiguračního XML souboru.

Objekty přistupující k datům v naší aplikaci

V aplikaci byly vytvořeny tyto DAO mapující jednotlivé tabulky databáze:

DAO	**tabulka v databázi**
`logEntryCouplingDao`	`logvazba`
`logEntryDao`	`log`
`logEntryParameterDao`	`log_promenna`
`messageDao`	`zprava`
`messageParameterDao`	`promenna`
`parameterTypeDao`	`typ_promenna`

Každý DAO je vytvořen jako rozhraní. Například `MessageDAO` (soubor `MessageDao.java`):

```
import cz.aegis.util.log.model.Message;
import cz.aegis.util.log.model.MessageIdentifier;

public interface MessageDao extends GenericDao<Message> {

	Message findByMessageSourceAndCode(String messageSource,String code);

	Message findByMessageIdentifier(MessageIdentifier messageIdentifier);
    }
```

Soubor `Message.java` pak obsahuje konkrétní parametry zprávy a další metody. Důležitá je především metoda pro vytvoření unikátního identifikátoru zprávy, kterým je dvojice kód zprávy a zdroj:

```
public class Message extends Entity {

	public static final String TABLE = "ZPRAVA";

	private String messageSource;

	private String messageCode;

	private Boolean activity = Boolean.TRUE;

	private Integer messageSeverity;

	private String messageText;

public void setMessageIdentifier(MessageIdentifier messageIdentifier) {
		if (messageIdentifier != null) {
			messageSource = messageIdentifier.getMessageSource();
			messageCode = messageIdentifier.getMessageCode();
		} else {
			messageSource = null;
			messageCode = null;
		}
	}

	public MessageIdentifier getMessageIdentifier() {
		return MessageIdentifier.valueOf(messageSource, messageCode);
	}

    .....
```

Aplikační kontext ve Springu

Spring Context je v naší aplikaci vytvořen konfiguračním souborem `businessLogger-appContext.xml`. Ten obsahuje jednotlivé elementy (*beany*) reprezentující objekty. Každá *beana* má vlastní jméno, pomocí kterého se na ni odkazují ostatní beany v definici závislostí jednotlivých bean na sobě v tagu `property` (konkrétní propojení objektů můžeme vidět na obrázcích 12 a 13). Tím je v konfiguračním souboru zajištěno vytvoření vazeb mezi jednotlivými objekty uvnitř aplikace.

```
<bean id="businessLogger"
class="cz.aegis.util.log.impl.BusinessLoggerImpl">
      <property name="logEntryDao" ref="logEntryDao" />
      <property name="logEntryParameterDao" ref="logEntryParameterDao" />
      <property name="messageDao" ref="messageDao" />
      <property name="messageParameterDao" ref="messageParameterDao" />
      <property name="parameterTypeDao" ref="parameterTypeDao" />
      <property name="logEntryCouplingDao" ref="logEntryCouplingDao" />
</bean>

<bean id="businessLoggerManager"
class="cz.aegis.util.log.impl.BusinessLoggerManagerImpl">
      <property name="messageDao" ref="messageDao" />
      <property name="messageParameterDao" ref="messageParameterDao" />
      <property name="parameterTypeDao" ref="parameterTypeDao" />
</bean>

<bean id="messageDao" class="cz.aegis.util.log.dao.jdbc.MessageDaoJdbc">
      <property name="dataSource" ref="dataSource" />
      <property name="dataFieldMaxValueIncrementer"
ref="businessLoggerIdIncrementer" />
</bean>

<bean id="parameterTypeDao"
class="cz.aegis.util.log.dao.jdbc.ParameterTypeDaoJdbc">
      <property name="dataSource" ref="dataSource" />
      <property name="dataFieldMaxValueIncrementer"
ref="businessLoggerIdIncrementer" />
</bean>
```

.....

Použití rozhraní `dataSource` je alternativou k získávání spojení z databáze pomocí objekt `Connection`. V našem případě je objekt `dataSource` součástí `businessLogger-appContext-test.xml`

```
<!-- importujeme definice v hlavnim aplikacnim kontextu -->
      <import resource="classpath:businessLogger-appContext.xml" />
      <context:property-placeholder
location="classpath*:database.properties.test" />
      <bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName"
            value="${jdbc.driverClassName}" />
            <property name="url" value="${jdbc.url}" />
            <property name="username" value="${jdbc.username}" />
            <property name="password" value="${jdbc.password}" />
      </bean>
```

.....

Konkrétní použité nastavitelné hodnoty pro spojení s databází jsou nastaveny v souboru `database.properties.test`.

```
jdbc.driverClassName=net.sourceforge.jtds.jdbc.Driver
jdbc.url=jdbc:jtds:sqlserver://..........
jdbc.username=.....
jdbc.password=.....
```

Spring JDBC framework

Z dalších možností, které Spring JDBC poskytuje, byly použití třídy `JdbcTemplate`, `JdbcDaoSupport` a rozhraní `RowMapper`.

Třída `JdbcDaoSupport` je abstraktní třídou, která zabezpečuje instance `JdbcTemplate`. Třída `JdbcTemplate` slouží k vykonávání SQL příkazů pomocí návrhového vzoru `Template method`. Ten zajišťuje, že nemusíme pro každý SQL dotaz psát do kódu stále stejné kroky (získání databázového spojení, vytvoření statementu, provedení SQL příkazu, uzavření `ResultSet`, `Statement`, `Connection`).

Příklad použití SQL dotazu v za použití třídy `JdbcDaoSupport` můžeme nalézt v souboru MessegeDaoJdbc.java, kde provádíme uložení parametrů zprávy do databáze.

```
public class MessageDaoJdbc extends SimpleJdbcDaoSupport implements
MessageDao {

	private DataFieldMaxValueIncrementer dataFieldMaxValueIncrementer;

	private static final String INSERT_SQL = "insert into"
			+ " zprava(zprava_id, zdroj, kod, zavaznost, aktivita,
			text)"
			+ " values (?, ?, ?, ?, ?, ?)";

	.........

	public Message insert(Message message) {
		Assert.notNull(message);

		// ziskame dalsi id z databaze
		Long id = dataFieldMaxValueIncrementer.nextLongValue();
		message.setId(id);

		// dame priznak aktivity jako text
		String activity =
	ActivityTranslator.getAsString(message.getActivity());

		getSimpleJdbcTemplate().update(
			INSERT_SQL,
			new Object[] {
					message.getId(),
					message.getMessageSource(),
					message.getMessageCode(),
					message.getMessageSeverity(),
					activity, message.getMessageText() });
		return message;
	}
```

Rozhraní `Rowmaper` prochází `ResultSet` a z jednotlivých řádků `ResultSetu` vytváří doménové objekty, které DAO vrací.

```
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.simple.ParameterizedRowMapper;

import cz.aegis.util.log.model.Message;

public class MessageRowMapper implements
 ParameterizedRowMapper<Message> {

 public Message mapRow(ResultSet rs, int rowNum) throws SQLException {
       Message message = new Message();

       message.setId(rs.getLong("zprava_id"));
       message.setMessageSource(rs.getString("zdroj"));
       message.setActivity(ActivityTranslator.getFromResultSet(rs,
       "aktivita"));
       message.setMessageCode(rs.getString("kod"));
       message.setMessageSeverity(rs.getInt("zavaznost"));
       message.setMessageText(rs.getString("text"));

       return message;
 }

}
```

9.1.5 Vytváření záznamů v logu

Vytvoření záznamu v logu zajišťuje rozhraní `businesLogger` a třída, která ho implementuje `businesLoggerImpl`. Záznam v logu je vytvářen pomocí jedinečného identifikátoru zprávy, kterým je dvojice kód zprávy a zdroj, dále obsahuje seznam objektů, ke kterým se daný záznam váže (logované doménové objekty) a hodnoty parametrů, které reprezentují hodnoty proměnných předdefinovaných zpráv. Konkrétní hodnoty jsou uloženy do tabulky `log` a `log_promenna`. Příslušné metody v `businesLoggerImpl.java`:

```
public LogEntryWMessage createLogEntry(MessageIdentifier
messageIdentifier, List<Object> parameters) {

.....
```

Následně je vytvořena vazba na doménový objekt s identifikací (`tgt_id`, `tgt_table`) a vytvořen záznam v tabulce `logvazba`.

```
public List<LogEntryCoupling> addCouplings(LogEntry logEntry,
                List<ExternalObjectIdentification> identifications) {
.....
```

9.1.6 Vytváření zpráv a proměnných

Logovací systém také umožňuje pomocí navržených tříd vytvořit konkrétní předdefinovanou zprávu spolu s uložením proměnných a jejich typů. To zajišťuje `businessLoggerManagerImpl` pomocí metod:

```
public void createMessage(final Message message,
                final List<ParameterType> parameters) {
            Assert.notNull(message);

            messageDao.insert(message);

            if (parameters != null) {
                  for (int i = 0; i < parameters.size(); i++) {
                        createMessageParameter(message, parameters.get(i),
                        i);
                  }
            }
      }

      private MessageParameter createMessageParameter(Message message,
                  ParameterType parameterType, int index) {

            if (parameterType.getId() == null) {
                  // vlozime parameter type do databaze
                  parameterTypeDao.insert(parameterType);
            }

            MessageParameter mp = new MessageParameter();

            mp.setMessageId(message.getId());
            mp.setOrderNumber(index);
            mp.setParameterTypeId(parameterType.getId());

            messageParameterDao.insert(mp);

            return mp;
      }
     .....
```

9.2 Testování

9.2.1 Testy aplikace

Test byl spuštěn jako JUnit test v Eclipse Platform 3.4.0. testovacích tříd v adresáři `src/test/java`.

Vložení konkrétního testovacího záznamu do logu a do vazební tabulky v databázi je vytvořeno v `BusinessLoggerTest.java`.

Vložení konkrétních testovacích zpráv a proměnných do databáze je vytvořeno v `BusinessLoggerTest.java`.

Testovací data, která aplikace pro uložení záznamů do databáze používá jsou v souboru `businessLoger-test-data.sql` v adresáři `src/test/resources`.

Zobrazení výsledných testovacích záznamů v databázi je součástí přílohy 2.

9.2.2 Integrace do CRÚ

V další fázi testování systému bylo původním záměrem, pro otestování jeho funkčnosti nad konkrétními daty, začlenění logovacího systému do aplikace Centrálního registru úvěrů (CRÚ), na kterém firma pracuje. V této aplikaci by vytvořený logovací systém zajišťoval logování vybraných doménových objektů.

Centrální registr úvěrů je aplikace, která zpracovává informace od konkrétních bank o klientech a jejich pohledávkách.

V CRÚ je každý klient doménovým objektem s atributy (jako například rodné číslo `RC`, ičo `ICO`, `ULICE` a další). Samotné atributy klienta (`RC`, `ICO`, `ULICE`..) jsou pak také uloženy jako doménové objekty s vlastními atributy (`HODNOTA`, `POPIS`...). Aplikace také obsahuje další doménové objekty reprezentující informace týkající se především pohledávek.

Každý doménový objekt má `id`, které je jedinečné v rámci celé aplikace CRÚ (tento systém používá firma i ve svých dalších aplikacích). To je zajištěno prostřednictvím klíče, který, kromě dalších informací, uchovává poslední vytvořenou hodnotu `id` jakéhokoliv doménového objektu v rámci celé aplikace. Při vytváření dalšího záznamu v libovolném doménovém objektu je hodnota z klíče inkrementována, přiřazena nově vzniklému záznamu a uložena do klíče jako poslední vytvořené `id`.

Po připojení logovacího systému by jedinečné `id` v rámci aplikace logovaného doménového objektu bylo jako `tgt_id` ukládáno do tabulky `logvazba`. Sloupec `tgt_table` by pak obsahoval název tabulky doménového objektu (`KLIENT`, `POHLEDAVKA`, `ULICE`, `ICO` a podobně).

Pro účel logování doménových objektů aplikace CRÚ byly do databáze firmy uloženy konkrétní předdefinované zprávy (soubor `Cru_Message.sql` a `Aeg_Cru_Message.sql` v adresáři `database` na přiloženém CD). Testování logovacvího systému v této fázi však nebylo již z časových důvodů reálně uskutečněno. Vytvořené zprávy pro CRÚ byly použity jako testovací data spolu s dalšími vytvořenými testovacími záznamy (soubor `Test_data.sql`).

9.3 Práce se záznamy v logu

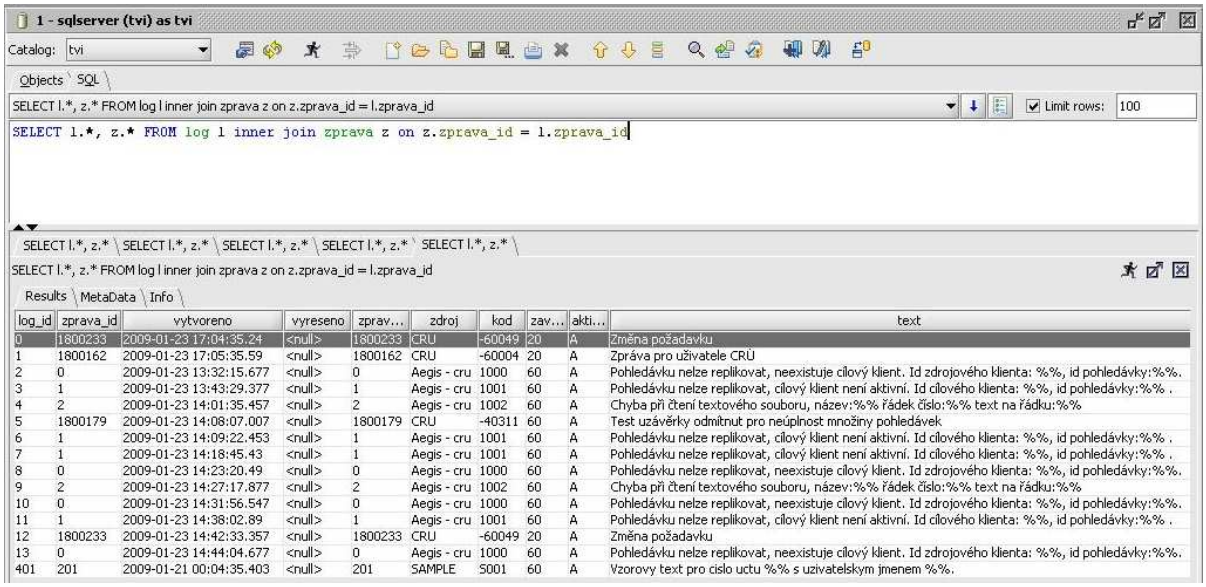
Jak již bylo řečeno, logovací systém neobsahuje prezentační vrstvu pro zobrazování záznamu v logu, vyhledávání v logu a editaci dat. To provádí uživatel (zaměstnanec firmy), který má přístup přímo k databázi, pomocí SQL příkazů v prostředí SQL klienta. K tomuto účelu je ve firmě využíván Squirel SQL klient Version 2.6.8.

Dotazování nad záznamy v logu v databázi umožňuje vyhledávat záznamy podle konkrétních kritérií. Například podle úrovně závažnosti, zdroje, času, podle hodnot parametrů, nebo záznamy konkrétního doménového objektu.

Jako příklad uveďme několik konkrétních SQL dotazů realizovaných nad testovacími daty spolu s výstupy v podobě snímků obrazovky ze Squirel SQL klienta po jejich provedení.

Dotaz, který vrátí všechny záznamy v logu a příslušné předdefinované zprávy.

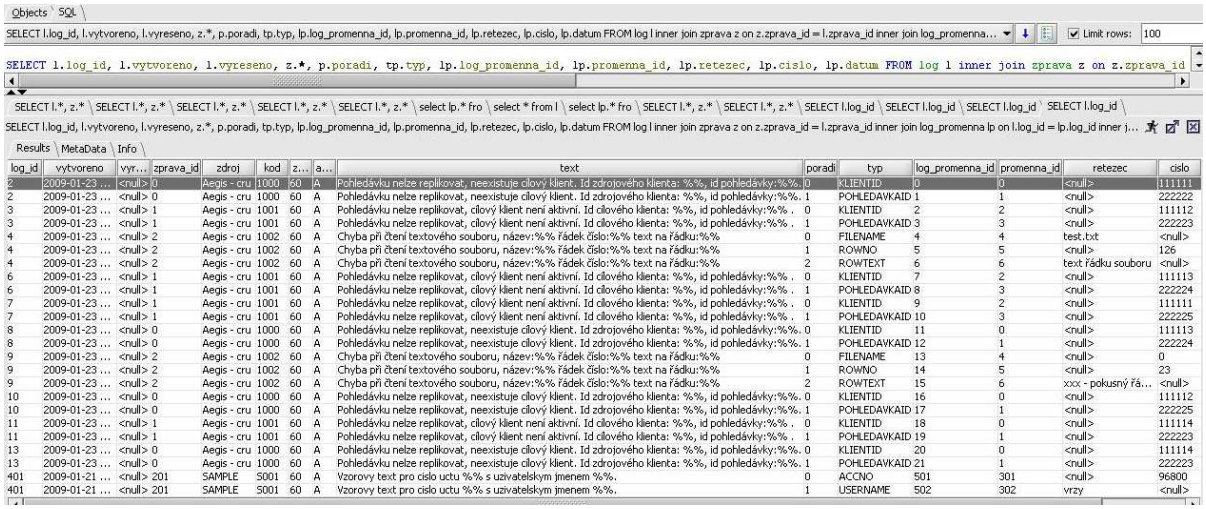
```
SELECT l.*, z.* FROM log l inner join zprava z on z.zprava_id = l.zprava_id
```



log_id	zprava_id	vytvoreno	vyreseno	zprav...	zdroj	kod	zav...	akti...	text
0	1800233	2009-01-23 17:04:35.24	<null>	1800233	CRU	-60049	20	A	Změna požadavku
1	1800162	2009-01-23 17:05:35.59	<null>	1800162	CRU	-60004	20	A	Zpráva pro uživatele CRÚ
2	0	2009-01-23 13:32:15.677	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
3	1	2009-01-23 13:43:29.377	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
4	2	2009-01-23 14:01:35.457	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%
5	1800179	2009-01-23 14:08:07.007	<null>	1800179	CRU	-40311	60	A	Test uzávěrky odmítnut pro neúplnost množiny pohledávek
6	1	2009-01-23 14:09:22.453	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
7	1	2009-01-23 14:18:45.43	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
8	0	2009-01-23 14:23:20.49	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
9	2	2009-01-23 14:27:17.877	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%
10	0	2009-01-23 14:31:56.547	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
11	1	2009-01-23 14:38:02.89	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
12	1800233	2009-01-23 14:42:33.357	<null>	1800233	CRU	-60049	20	A	Změna požadavku
13	0	2009-01-23 14:44:04.677	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
401	201	2009-01-21 00:04:35.403	<null>	201	SAMPLE	S001	60	A	Vzorovy text pro cislo uctu %% s uzivatelskym jmenem %%.

Dotaz, který vrátí všechny záznamy v logu s příslušnými předdefinovanými zprávami s parametry, hodnotami parametrů, jejich pořadím, identifikací proměnných a jejich typy.

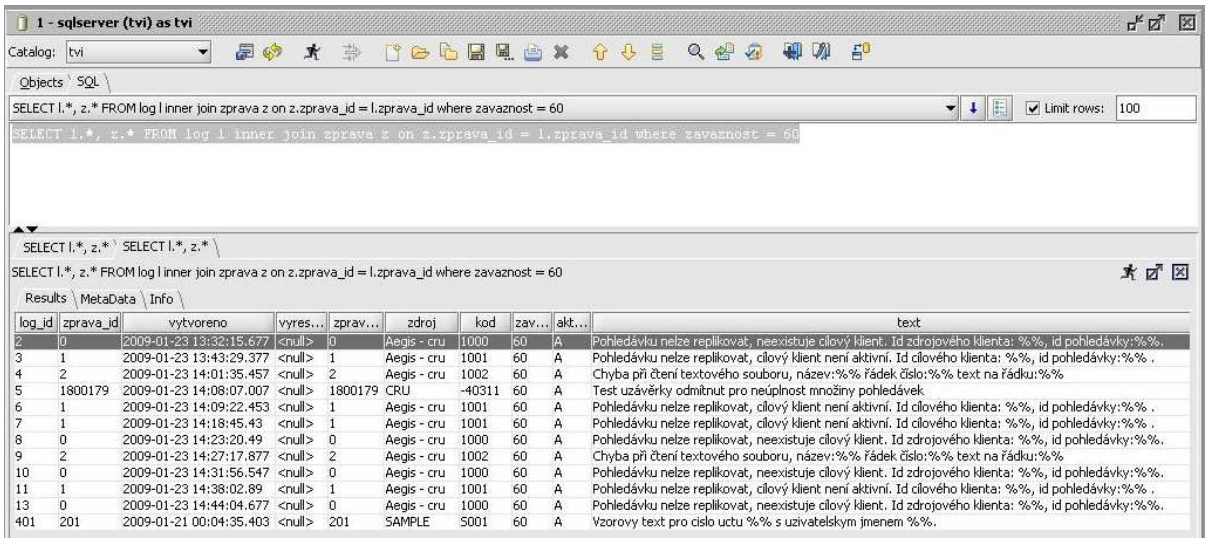
```
SELECT l.log_id, l.vytvoreno, l.vyreseno, z.*, p.poradi, tp.typ,
lp.log_promenna_id, lp.promenna_id, lp.retezec, lp.cislo, lp.datum
FROM log l inner join zprava z on z.zprava_id = l.zprava_id  inner
join log_promenna lp on l.log_id = lp.log_id inner join promenna p
on p.promenna_id = lp.promenna_id inner join typ_promenna tp on
tp.typ_promenna_id = p.typ_promenna_id
```



log_id	vytvoreno	vyr...	zprava_id	zdroj	kod	z...	a...	text	poradi	typ	log_promenna_id	promenna_id	retezec	cislo
2	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	0	KLIENTID	0	0	<null>	111111
2	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	1	POHLEDAVKAID	1	1	<null>	222222
3	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	0	KLIENTID	2	2	<null>	111112
3	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	1	POHLEDAVKAID	3	3	<null>	222223
4	2009-01-23 ...	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%	0	FILENAME	4	4	test.txt	<null>
4	2009-01-23 ...	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%	1	ROWNO	5	5	<null>	126
4	2009-01-23 ...	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%	2	ROWTEXT	6	6	text řádku souboru	<null>
6	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	0	KLIENTID	7	2	<null>	111113
6	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	1	POHLEDAVKAID	8	3	<null>	222224
7	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	0	KLIENTID	9	2	<null>	111111
7	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	1	POHLEDAVKAID	10	3	<null>	222225
8	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	0	KLIENTID	11	0	<null>	111113
8	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	1	POHLEDAVKAID	12	1	<null>	222224
9	2009-01-23 ...	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%	0	FILENAME	13	4	<null>	0
9	2009-01-23 ...	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%	1	ROWNO	14	5	<null>	23
9	2009-01-23 ...	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%	2	ROWTEXT	15	6	xxx - pokusný řá...	<null>
10	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	0	KLIENTID	16	0	<null>	111112
10	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	1	POHLEDAVKAID	17	1	<null>	222225
11	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	0	KLIENTID	18	0	<null>	111114
11	2009-01-23 ...	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .	1	POHLEDAVKAID	19	1	<null>	222223
13	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	0	KLIENTID	20	0	<null>	111114
13	2009-01-23 ...	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.	1	POHLEDAVKAID	21	1	<null>	222223
401	2009-01-21 ...	<null>	201	SAMPLE	S001	60	A	Vzorovy text pro cislo uctu %% s uzivatelskym jmenem %%.	0	ACCNO	501	301	<null>	96800
401	2009-01-21 ...	<null>	201	SAMPLE	S001	60	A	Vzorovy text pro cislo uctu %% s uzivatelskym jmenem %%.	1	USERNAME	502	302	vrzy	<null>

Dotaz, který vrátí všechny záznamy v logu a příslušné předdefinované zprávy s úrovní závažnosti 60.

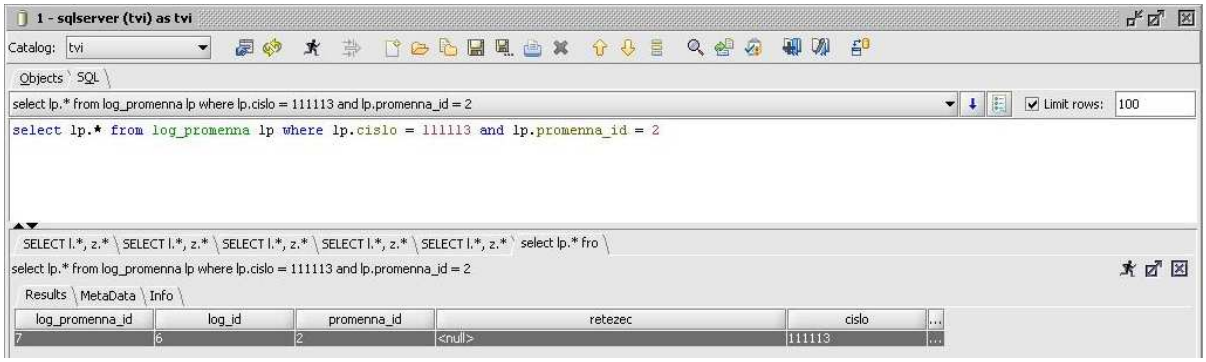
```
SELECT l.*, z.* FROM log l inner join zprava z on z.zprava_id =
l.zprava_id where zavaznost = 60
```



log_id	zprava_id	vytvoreno	vyres...	zprav...	zdroj	kod	zav...	akt...	text
2	0	2009-01-23 13:32:15.677	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
3	1	2009-01-23 13:43:29.377	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
4	2	2009-01-23 14:01:35.457	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%
5	1800179	2009-01-23 14:08:07.007	<null>	1800179	CRU	-40311	60	A	Test uzávěrky odmítnut pro neúplnost množiny pohledávek
6	1	2009-01-23 14:09:22.453	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
7	1	2009-01-23 14:18:45.43	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
8	0	2009-01-23 14:23:20.49	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
9	2	2009-01-23 14:27:17.877	<null>	2	Aegis - cru	1002	60	A	Chyba při čtení textového souboru, název:%% řádek číslo:%% text na řádku:%%
10	0	2009-01-23 14:31:56.547	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
11	1	2009-01-23 14:38:02.89	<null>	1	Aegis - cru	1001	60	A	Pohledávku nelze replikovat, cílový klient není aktivní. Id cílového klienta: %%, id pohledávky:%% .
13	0	2009-01-23 14:44:04.677	<null>	0	Aegis - cru	1000	60	A	Pohledávku nelze replikovat, neexistuje cílový klient. Id zdrojového klienta: %%, id pohledávky:%%.
401	201	2009-01-21 00:04:35.403	<null>	201	SAMPLE	S001	60	A	Vzorovy text pro cislo uctu %% s uzivatelskym jmenem %%.

Dotaz, který vrátí všechny záznamy v logu, kde konkrétní proměnná (`id_promenne = 2`) obsahuje hodnotu 111113.

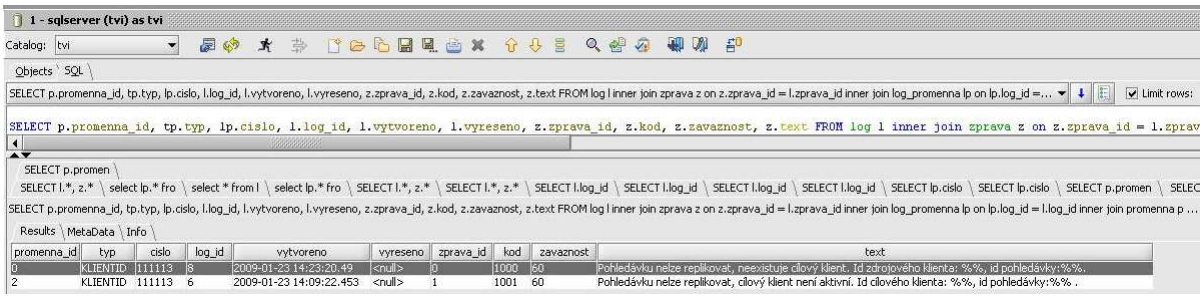
```
SELECT lp.* from log_promenna lp where lp.cislo = 111113 and
lp.promenna_id = 2
```



log_promenna_id	log_id	promenna_id	retezec	cislo
7	6	2	<null>	111113

Dotaz, který vrátí všechny záznamy v logu konkrétního klienta (typ proměnné KLIENTID) s číslem 111113.

```
SELECT p.promenna_id, tp.typ, lp.cislo, l.log_id, l.vytvoreno,
l.vyreseno, z.zprava_id, z.kod, z.zavaznost, z.text FROM log l inner
join zprava z on z.zprava_id = l.zprava_id inner join log_promenna
lp on lp.log_id = l.log_id  inner join promenna p on p.zprava_id =
z.zprava_id inner join typ_promenna tp on tp.typ_promenna_id =
p.typ_promenna_id where lp.cislo = 111113 and tp.typ = 'KLIENTID'
```



10 Možnosti rozšíření

Realizace prezentační vrstvy:

Možností dalšího rozšíření logovacího systému je především vytvoření prezentační vrstvy v podobě grafického uživatelského rozhraní, které bude poskytovat komfortnější práci s logem, než pomocí SQL příkazů. Funkce, které by mělo poskytovat je především vizualizace záznamů logu a možnost vyhledávání (selekce) na základě parametrů jakými mohou být například hodnoty jednotlivých proměnných zprávy v logu. Součástí prezentační vrstvy by také měla být z bezpečnostních důvodů autorizace přístupu k logu.

Nasazení v konkrétní aplikaci:

Nasazení systému do „ostrého“ logování v nějaké konkrétní aplikaci a připojení k reálným doménovým objektům.

Filtrace ukládaných záznamů:

Dalším rozšířením by byla možná implementace filtrů, které by zajišťovaly, že do logu nebudou ukládány všechny zprávy, ale pouze ty, které budou splňovat příslušná filtrační kritéria (například úroveň závažnosti), jako je tomu u existujících logovacích systémů, které zde byly popsány.

Jazykové variace předdefinovaných zpráv:

Bylo by možné vytvořit různé jazykové varianty předdefinovaných zpráv, přičemž změna jazyka zpráv by byla možná pouhou jednoduchou změnou v mapovaní databázové tabulky.

11 Závěr

Tato diplomová práce se zabývala analýzou systému IBM AS/400 a jeho systému zpráv, na základě jehož některých principů měl být podle zadání firmy Aegis s.r.o. navržen a následně implementován v jazyce Java vlastní logovací systém pro vytváření logovacích záznamů k doménovým objektům v aplikacích.

V práci byl nejprve analyzován AS/400 a jeho systém zpráv. Dalším bodem byla, pro zjištění možností a principů používaných pro logování, analýza dalších existujících logovacích systémů, přičemž zde byly popsány syslog, syslog-ng, Log4j.

Následně byly na základě analogie systému zpráv na AS/400 definovány obecné vlastnosti logovacího systému.

Dalším bodem práce byl pak samotný návrh logovacího systému a jeho implementace, které se práce věnuje ve své druhé polovině. Na základě požadavků zadavatele byl vytvořen návrh databáze a následně v jazyce Java, za použití aplikačního rámce Spring a rozhraní pro přístup k databázi JDBC, realizován vlastní logovací systém. Jeho funkčnost byla otestována vzorkem testovacích dat.

Výhodami vytvořeného logovacího systému je jeho obecnost (nezávislost na operačním systému i na konkrétní aplikaci) a možnost vyhledávání v logu na základě konkrétních parametrů. To může být využitelné například při sledování činnosti konkrétních uživatelů a obecně v možnosti provádění další analýzy logu.

Podmínkou fungování logovacího systému je, že jednotlivé logované doménové objekty musí mít unikátní `id` v rámci celé aplikace (nikoliv pouze v rámci tabulky).

Nevýhodou systému je, že jeho součástí není prezentační vrstva pro vizualizaci a snadnější práci s logem, což je jednou z možností jeho rozšíření.

Použitá literatura:

[1] Aegis s.r.o., Na Pankráci 58, 140 00 Praha 4: *Historie a charakteristika AS/400*, Firemní materiál, 1999.

[2] Aegis s.r.o., Na Pankráci 58, 140 00 Praha 4: *Jazyk CL*, Firemní materiál, 1999.

[3] Aegis s.r.o., Na Pankráci 58, 140 00 Praha 4: *Základy práce v prostředí AS/400*, Firemní materiál, 1999.

[4] Aegis s.r.o., Na Pankráci 58, 140 00 Praha 4: *Zprávy na AS/400*, Firemní materiál, 1999.

[5] *Collecting syslog messages into an SQL database with syslog-ng Premium Edition*, BalaBit IT Security: 2008. Dokument dostupný na URL: http://www.balabit.hu/dl/white_papers/syslog-ng-v2.1-whitepaper-syslog-into-sql-database-en.pdf (leden 2008).

[6] Brůha, Lubomír: *Java: Hotová řešení*, Computer Press: 2003.

[7] Deveriya, Anand: *An Overview of the syslog Protocol*, in *Network Administrators Survival Guide*, Cisco Press: 2005. Dokument dostupný na URL: http://www.ciscopress.com/content/images/1587052113/samplechapter/1587052113content.pdf (leden 2009).

[8] Faťun, Martin: *Patří AS/400 do starého železa?*, in *Computerworld: Deník pro IT profesionály*, 2000. Dokument dostupný na URL: http://archiv.computerworld.cz/cwarchiv.nsf/clanky/B254B1F20F91217FC12569B00056DC10?OpenDocument (listopad 2008).

[9] Goyal, Vikram: *Build Flexible Logs With log4j*, 2002. Dokument dostupný na URL: http://www.onjava.com/pub/a/onjava/2002/08/07/log4j.html (leden 2009)

[10] Haring, David: *Analýza systémových logů: Logcheck*, in *Linuxové noviny*, 01-02/2001. Dokument dostupný na URL: http://www.linux.cz/noviny/2001-02/clanek03.html (leden 2008).

[11] Haring, David: *Jak na systémový log?*, in *Linuxové noviny*, 03-04/2001. Dokument dostupný na URL: http://www.linux.cz/noviny/2001-04/clanek03.html (leden 2008).

[12] *IBM AS/400 Information security*. Dokument dostupný na URL: www.auditnet.org/docs/AS400%20Information%20Security%20APG.doc (listopad 2008)

[13] IBM Corporation, *iSeries CL Programming*, Version 5, IBM: 2002. Dokument dostupný na URL: http://publib.boulder.ibm.com/infocenter/iseries/v5r3/topic/books/sc415721.pdf (leden 2009).

[14] *IBM System i* in *Wikipedia: The free encyklopedia* (En), 2008. Dokument dostupný na URL: http://en.wikipedia.org/wiki/IBM_System_i (listopad 2008).

[15] Janeček, Martin, Jirků, František: *Je čtyřstovka ohroženým druhem?* in *The Blue Rose*, 1/2006, IBM ČR, Dokument dostupný na URL: http://www-05.ibm.com/cz/think/download/bluerose0106.pdf (listopad 2008).

[16] Klobasa, Pavel: *Log4J - logujeme s přehledem*, 2008. Dokument dostupný na URL: http://vyvojari.oxyonline.cz/log4j-logujeme-s-prehledem (leden 2009).

[17] Koudelka, Pavel: *Historie operačních systémů*. Dokument dostupný na URL: http://airborn.webz.cz/histos.html (listopad 2008).

[18] Machacek, Jan, ed.: *Pro Sping 2.5*, Apress: 2008.

[19] Novotný, Róbert: *Nadišla jar v krajine DAO (Java)*, 2008. Dostupný na URL: http://ics.upjs.sk/~novotnyr/wiki/Java/SpringAJDBC (leden 2009)

[20] Pichlík, Roman: *Spring Framework - představení J2EE lightweight kontejneru*, 2005. Dokument dostupný na URL: http://interval.cz/clanky/spring-framework-predstaveni-j2ee-lightweight-kontejneru/ (leden 2009)

[21] Rose, Thornton: *Logging in Java Applications*. Dokument dostupný na URL: http://www.developer.com/java/other/article.php/1404951 (leden 2009)

[22] *Regulatory compliance and system logging***,** BalaBit IT Security: 2008. Dokument dostupný na URL: https://www.balabit.com/dl/white_papers/syslog-ng-v3.0-whitepaper-compliance-en.pdf (prosinec 2008).

[23] Stein, René: *Návrh aplikací v jazyce UML - Unified Modeling Language*, 2003. Dokument dostupný na URL: http://interval.cz/clanky/navrh-aplikaci-v-jazyce-uml-unified-modeling-language/ (leden 2009)

[24] *Syslog-ng: secure log collection, processing and storage***,** BalaBit IT Security: 2008. Dokument dostupný na URL: http://www.balabit.com/dl/brochures/syslog-ng-v3.0-flyer-en.pdf (prosinec 2008).

[25] Šeda, Jan: *Úvod do JDBC*, 2003. Dokument dostupný na URL: http://interval.cz/clanky/uvod-do-jdbc/ (leden 2009)

[26] Šenk, Zdeněk: *Technologie pro perzistenci objektů v Javě*, Diplomová práce: Vysoké učení technické v Brně FIT: 2007.

[27] Štrauch, Adam: *Syslog-ng: pořádek v log souborech*, 2008. Dokument dostupný na URL: http://www.root.cz/clanky/syslog-ng-poradek-v-log-souborech/ (leden 2009).

[28] *The syslog-ng 3.0 Administrator Guide***,** BalaBit IT Security: 2008. Dokument dostupný na URL: https://www.balabit.com/dl/guides/syslog-ng-v3.0-guide-admin-en.pdf (prosinec 2008).

[29] Thon, Ladislav: Spring Web MVC framework, 2006. Dokument dostupný na URL: http://www.kiv.zcu.cz/~brada/vyuka/files/pia/ppp/spring/ (leden 2009)

[30] Tichý, Jan: *Logování událostí*, in *Programová podpora tvorby webových aplikací*, Diplomová práce: Vysoká škola ekonomická v Praze: 2004. Dokument dostupný na URL: http://www.jantichy.cz/diplomka/pozadavky/logovani (prosinec 2008).

[31] Tišnovský, Pavel: *Nástroje pro tvorbu UML diagramů*, 2005. Dokument dostupný na http://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu/ (leden 2009)

[32] www stránky *IBM iSeries Information Center v5r4*. URL: http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=/rzahg/icmain.htm (leden 2009)

Příloha 1

Postup vytvoření databáze a konfigurace systému:

Databáze:
Aplikace je navržena pro práci s MS SQL nebo Postgres databází.

Vygenerování create scriptu pro vytvoření databáze:
Ve Visual Paradigm for UML 6.4 Enterprise Edition je vytvořen er-diagram (soubor `log_system.vpp`), ze kterého lze vygenerovat (generate sql) create script pro MS Sql nebo Postgres databázi.

Tabulka log_sequence:
Dále je nutné vytvořit tabulku log_sequence, která zajišťuje inkrementaci id při vytváření záznamů ve všech tabulkách databáze.

Pro MS SQL:
```
create table log_sequence (id bigint identity);
insert into log_sequence default values;
```

pro PostgreS:
```
create sequence log_sequence start with 1000;
```

database.properties.test:
Dále je nutné změnit nastavení připojení k databázi v souboru database.properties.test

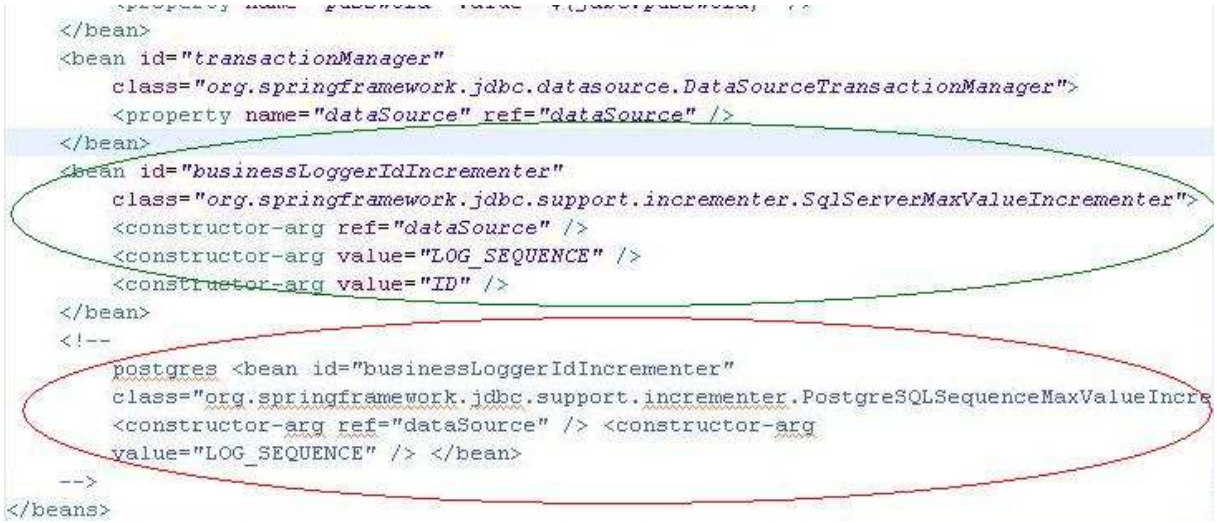
PostgreS:

jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://.....
jdbc.username=.....
jdbc.password=.....

MS SQL:

jdbc.driverClassName=net.sourceforge.jtds.jdbc.Driver
jdbc.url=jdbc:jtds:sqlserver://.....
jdbc.username=.....
jdbc.password=.....

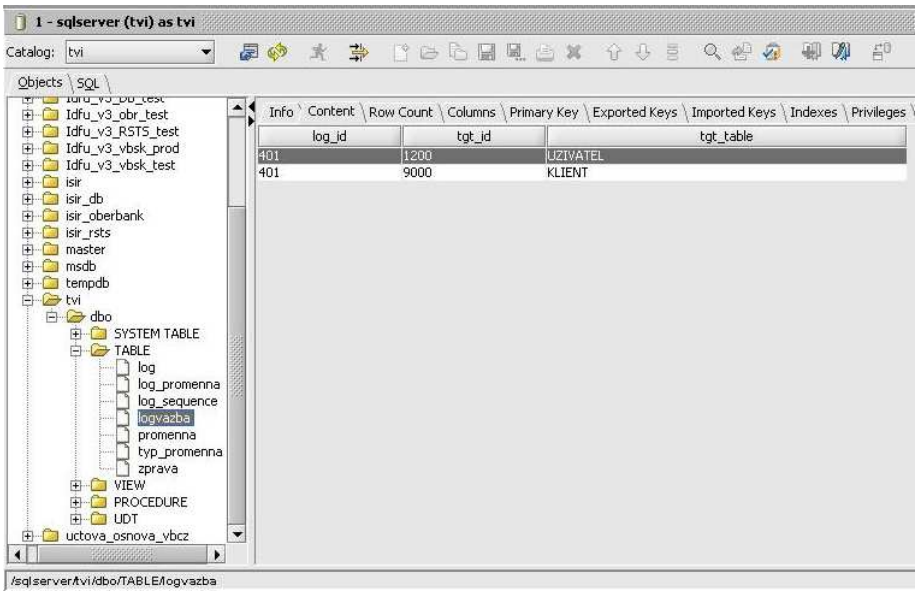
businessLogger-appContext.xml:
Pokud je použit Postgres, je v tomto souboru třeba odkomentovat beanu pro Postgres a zakomentovat beanu pro MS SQL.


</bean>
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource" ref="dataSource" />
</bean>
<bean id="businessLoggerIdIncrementer"
class="org.springframework.jdbc.support.incrementer.SqlServerMaxValueIncrementer">
<constructor-arg ref="dataSource" />
<constructor-arg value="LOG_SEQUENCE" />
<constructor-arg value="ID" />
</bean>
<!--
postgres <bean id="businessLoggerIdIncrementer"
class="org.springframework.jdbc.support.incrementer.PostgreSQLSequenceMaxValueIncre
<constructor-arg ref="dataSource" /> <constructor-arg
value="LOG_SEQUENCE" /> </bean>
-->
</beans>

Příloha 2

Výsledky testu v podobě záznamů v databázi po spuštění testovacích tříd aplikace.

Testovací záznamy v tabulce `logvazba`



Testovací záznamy v tabulce `log`

log_id	zprava_id	vytvoreno	vyreseno
401	201	2009-01-21 00:04:35.403	<null>

Testovací záznamy v tabulce `log_promenna`

log_promenna_id	log_id	promenna_id	retezec	cislo	datum
501	401	301	<null>	96800	00 00 00 00 0...
502	401	302	vrzy	<null>	00 00 00 00 0...

Testovací záznamy v tabulce `zprava`

zprava_id	zdroj	kod	zavaznost	aktivita	text
151	TEST	A010	60	A	Testovaci zprava, retezec '%%', datum %%, cislo %% a jeste jeden retezec stejneho typu %%.
201	SAMPLE	S001	60	A	Vzorovy text pro cislo uctu %% s uzivatelskym jmenem %%.

Testovací záznam v tabulce `log_sequence`

id
158

Testovací záznamy v tabulce promenna

promenna_id	zprava_id	typ_promenna_id	poradi
153	151	152	0
155	151	154	1
157	151	156	2
158	151	152	3
301	201	101	0
302	201	102	1

Testovací záznamy v tabulce typ_promenna

typ_promenna_id	zdroj	typ	datovy_...	popis
101	SAMPLE	ACCNO	LONG	Cislo uctu
102	SAMPLE	USERNAME	STRING	Uzivatelske jmeno
152	TEST	RETEZEC	STRING	<null>
154	TEST	DATUM	DATE	<null>
156	TEST	CISLO	LONG	<null>